Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

XML/XSLT systém pro lokální generování statických webových stránek
XML/XSLT System for Local Generation of Static Web Pages

2020 **Pavel Vavruška**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Pavel Vavruška**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: XML/XSLT systém pro lokální generování statických webových stránek
XML/XSLT System for Local Generation of Static Web Pages

Jazyk vypracování: čeština

Zásady pro vypracování:

Uživatel nemá vždy dostupný webový prostor současně s podporou generování jeho obsahu na straně serveru. Existují generátory statického obsahu webu, například Hyde založené na Pythonu, které takovou podporu nepotřebují. Umožňují přípravu webového obsahu na straně uživatele a jeho následné přenesení do cílového umístění. Snižuje to nároky na server a zvyšuje bezpečnost obsahu i uživatelských dat, která jsou zcela oddělena od webové prezentace.

1. Seznamte se s obdobně zaměřenými systémy, prostudujde související problematiku XML a XSLT transformací.
2. Proveďte podrobnou analýzu problému a navrhněte jeho řešení.
3. Navržený systém implementujte a zdokumentujte.
4. Navržené řešení porovnejte s jinými systémy.

Seznam doporučené odborné literatury:

[1] Systém Hyde: https://github.com/lakshmivyas/hyde
[2] XSLT http://www.w3schools.com/xsl/xsl_intro.asp

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. RNDr. Petr Šaloun, Ph.D.**

Datum zadání: 01.09.2019
Datum odevzdání: 30.04.2020

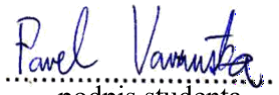
doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne: *15. května 2020*



podpis studenta

Poděkování

Rád bych poděkoval doc. Petrovi Šalounovi, Ph.D. za vedení mé bakalářské práce, taktéž za jeho veškerou pomoc, vstřícnost, konzultace a nepochybně cenné rady.

Abstrakt

Cílem této bakalářské práce je popsat standardy *XML* a *XSL transformace*, ze kterých vychází dnešní generování webových stránek. V teoretické části práce jsou popsány dvě základní skupiny webových stránek (dynamické a statické), včetně jejich vlastností. Práce se dále zaměřuje na tři základní jazyky pro tvorbu stránek statických – *HTML*, *CSS* a *JavaScript*. Pro tvorbu stránek jsou představeny možnosti jejich generování. Dynamické stránky se věnují zejména *Content Managment Systemu*, naopak u stránek statických jsou zmíněny základní vlastnosti generátorů. Je zde popsáno pět nejznámějších generátorů, přičemž největší pozornost je přenesena na generátor *Pelican*. V praktické části je vytvořen web dvěma možnostmi. První možnost využívá statický generátor *Pelican* a u druhé možnosti je pak zvoleno stahování stránek z internetu. Následuje úprava odkazů v *HTML* dokumentech, včetně nahrazování odkazů stažených ročníků konference.

Klíčová slova

XML; XSLT; World Wide Web; statické stránky; dynamické stránky; Pelican; HTML; CSS; JavaScript

Abstract

This bachelor's thesis aims to describe the *XML* and *XSL transformation* standards on which today's website generation is based. The theoretical part of the thesis describes two basic groups of websites (dynamic and static), including their properties. The work also focuses on three basic languages for creating static pages - *HTML*, *CSS* and *JavaScript*. The possibilities of generating pages are presented for the creation of pages. Dynamic pages are mainly devoted to the *Content Management System*, on the contrary, for static pages the basic properties of generators are mentioned. The five most well-known generators are described here, with the greatest attention being paid to the *Pelican* generator. In the practical part, the website is created with two options. The first option uses a static generator *Pelican* and the second option is to download pages from the Internet. This is followed by the modification of links in *HTML* documents, including the replacement of links downloaded from the conference years.

Key words

XML; XSLT; World Wide Web; static sites; dynamic sites; Pelican; HTML; CSS; JavaScript

Obsah

Seznam použitých zkratek

Zkratka	Význam
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
CERN	Evropská organizace pro jaderný výzkum
NCSA	National Center for Supercomputing Applications
SSG	Static Site Generator
OS	Operační systém
URL	Uniform Resource Locator
XML	eXtensible Markup Language
XSL	eXtensible Stylesheet Language

Seznam obrázků a tabulek

Seznam výpisů kódů

Úvod

V dnešní době se k získání a předávání informací využívá webových stránek. Tyto stránky jsou vytvářeny na základech *XML* a *XSL* transformací. Mnoho webů je generováno takovýmto způsobem. Máme celkem dvě možnosti: generování statického obsahu a generování dynamického obsahu. Abychom si zajistili zabezpečení a fungování stránek i bez použití webového serveru, je nejlepší cestou využití generování statického obsahu. Cílem této bakalářské práce je popsat samotné dynamické a statické stránky s jejich porovnáním, také ukázat možnosti generování těchto skupin stránek a v neposlední řadě přiblížit systém *World Wide Web*. V praktické části je pak hlavním cílem ukázat, jak můžeme statické stránky vytvořit pomocí lokálního generování statických webových stránek. Dílčím cílem je pak tvorba statických stránek, pomocí jejich stažení z internetu a následné statické propojení bez využití jakýchkoliv aktivních serverových vlastností.

První kapitola je věnována zmíněným standardům – *XML* a *XSLT*. Všechny dnešní webové stránky vycházejí právě z těchto dvou standardů. V průběhu této kapitoly je popsána charakteristika datového formátu *XML*, část jeho historie a také ukázka takového dokumentu. V části *XSL* je popsán samotný stylový jazyk *XSL*, ze kterého *XSLT* vychází, taktéž je zmíněn samotný vývoj *XSLT* a opět ukázán jeden takový stylový soubor.

Druhá kapitola je zaměřena na jednu z nejdůležitějších součástí dnešního internetu – *World Wide Web*. Je zde zmíněn jak vznik a vývoj *WWW*, tak také nejdůležitější technologie tohoto systému a také jeho nejdůležitější milníky. V neposlední řadě je zde vysvětleno samotné fungování *World Wide Webu* a taktéž porovnání takového systému s internetem, díky kterému máme přístup právě k webovým stránkám.

V třetí kapitole se zaobíráme samotnými webovými stránkami. Zmiňujeme zde dvě nejdůležitější skupiny – dynamické a statické stránky. U dynamických stránek popisujeme, jakým způsobem se vytváří samotný obsah a jaké jsou jejich typické vlastnosti. U statických stránek se zaměřujeme na jejich největší výhody, které se snažíme popsat a poté vybíráme nejvhodnější typy stránek pro využití statického webu. V druhé části se více zaměřujeme na samotné statické stránky. Jsou zde popsány tři nejzákladnější technologie pro vytváření takových webových stránek. Je zde popsána historie a vývoj s ukázkou dokumentů pro *HTML*, *CSS* a *JavaScript*.

Čtvrtá kapitola vysvětluje samotné možnosti vývoje těchto dvou skupin vytváření stránek a také je zde více rozveden využívaný generátor statických webových stránek *Pelican* s jeho vlastnostmi. V neposlední řadě zde nalezneme tabulku s porovnáním.

Poslední kapitola popisuje vytváření statických webových stránek. V prvním případě vysvětlujeme postup vytváření statických stránek za využití generátoru *Pelican*. Zaměříme se na instalaci potřebných součástí, vznik samotného obsahu a přidání vyhledávacího prvku. Druhá část popisuje, jakým způsobem můžeme získat web jakékoliv konference a vytvořit si svůj osobní archiv z jejich minulých ročníků. Aby však bylo možné takovýto archiv vytvořit, je potřeba stránky jakkoli získat (stáhnout či mít zálohu jiného ročníku na disku). Pokud weby budeme stahovat, musíme si z nich vybrat jejich nejdůležitější obsahové prvky a mít je uloženy ve statické podobě na disku. Pokud takovéto stránky máme, je potřebné je vzájemně propojit a upravit všechny odkazy do takové podoby, abychom k otevření stránek nepotřebovali webový server a my měli tak svou vygenerovanou lokální kopii statických stránek.

1 Technologie a standardy

Vytváření a generování dnešních statických webů funguje na principu *XML*/*XSLT*. Vždy existují zadaná data v jednom značkovacím jazyku (*XML*) a jsou graficky upravovány za využití stylových šablon (*XSLT*).

1.1 XML

XML je zkratka pro anglická slova *eXtensible Markup Language*, což se dá přeložit jako „rozšiřitelný značkovací jazyk“. Tento jazyk byl vyvinut a standardizován konsorciem *W3C*. Samotný jazyk vznikl v roce 1996, ale verze 1.0 byla standardizována až o dva roky později. Taktéž existuje verze *XML 1.1* (2001), ve které bylo hlavním úkolem zavést použití více znaků, jelikož verze 1.0 využívala pouze znaky *Unicode 2.0*. Znaky, které nejsou v *Unicode 2.0*, sice v *XML* 1.0 můžeme využít, ale nelze je použít v elementech a atributech. V tuto chvíli se tak využívají dvě verze *XML* – 1.0, která je v páté revizi, a 1.1, která si prošla druhou revizí. [1] [2]

Slovo Extensible nám v samotném názvu říká, že se jedná o rozšiřitelnost. Informace jsou totiž členěny do tagů podle logického uspořádání, a ne podle vzhledu – jako třeba *HTML*. Další slovo názvu je Markup, které zastupuje značkování. Jazyk *XML* vychází ze staršího standardu – *GML* (*Generalized Markup Language*). Zachovává se tedy síla *GML*, kterou je vytváření vlastních značek a univerzálnost. Také se dá říct, že *XML* dokument je platným *GML* dokumentem. Posledním slovem je Language (jazyk), které říká, že se jedná o formát, který je možné prohlížet, ale také editovat v různých editorech.

Již na začátku se počítalo, že v *XML* souboru se bude používat i jiný jazyk než angličtina. Začala se tedy používat znaková sada *ISO 10646*, která dokáže pojmout všechny znaky všech jazyků. Výhodou tedy je, že v souboru můžeme míchat jazyky a nejsme omezováni jen na jeden jazyk. V *XML* neexistují žádné předem definované značky, lze si je však zadefinovat v souboru *DTD* (*Document Type Definition*). Kontrolu validity dokumentu *XML* s námi zadanými značkami můžeme provést pomocí parseru. Další z možností, které *XML* nabízí, jsou odkazy, které mohou být vytvořeny jak v samotném dokumentu, tak i mezi dalšími dokumenty. Existují celkem tři možnosti – *XPath*, *XPointer* a *XLink*. *XPath* nám adresuje jednotlivé části dokumentu, *XPointer* je jeho nadstavbou a funguje na principu „zajímá mě x. odstavec y. kapitoly.“ Jazyk pro tvorbu odkazů se nazývá *XLink*, kdy se jednotlivé soubory určují podle *URL* adresy. [3]

1.1.1 XML dokument

Soubor *XML* se dá popsat jako jednoduchý textový soubor, který je tvořen znaky *Unicode*. Jak již bylo zmíněno, *XML* nemá za úkol popsat vzhled dokumentu, ale právě jeho obsah. Každý soubor *XML* začíná „<?“ a končí „?>“. V těchto značkách následně definujeme, že se jedná o *XML* dokument s určitou verzí a jeho znakovou sadou. Základním prvkem tohoto formátu jsou značky, které jsou párové. Značky jsou zapsány ve špičatých závorkách „<“ a „>“, kdy se k druhé značce páru přidá na jeho konec lomítko, „<“ zůstává stejná a končí se tedy „/>“. Cokoliv mezi těmito značkami se bere jako jejich obsah. Avšak značky se nesmí křížit, mohou se pouze vnořovat, což znamená, že před ukončením elementu, musí být ukončeny také jeho všechny vnitřní elementy. Taktéž je důležité poznamenat, že jeden *XML* soubor má pouze jeden kořenový element. Ukázka jednoduchého *XML* dokumentu se nachází ve výpisu kódu č. 1.

```
<?xml version="1.0" encoding="utf-8" ?>
<Osoby>
    <Osoba ID="1">
        <Jmeno>Pavel</Jmeno>
        <Prijmeni>Vavruška</Prijmeni>
        <Bydliste>
            <Mesto psc="73601">Havířov</Mesto>
            <Ulice>Hlavní třída</Ulice>
        </Bydliste>
    </Osoba>
<!-- Další osoby přidávat obdobně. -->
</Osoby>
```

Výpis kódu č. 1: *XML dokument*

1.2 XSL

XSL neboli *eXtensible Stylesheet Language* je univerzální stylový jazyk, který dokáže nabídnout veškeré funkce ostatních stylových jazyků a dále je rozšiřuje. Původně se sice mělo jednat o bratra *CSS* (*Cascading Style Sheets*), který řešil pouze vzhled. *XSL* již však umožňuje také generování obsahu nebo například číslování. Proto se v průběhu vývoje *XSL* verze 1.0 rozdělil jazyk na dva standardy – pro vzhled se využívaly *XSL – FO* (*XSL Formatting Objects*) a pro transformaci jazyka vznikl *XSLT* (*XSL Transformation*). [4]

1.2.1 XSLT

Standard *XSLT* byl přijat v roce 1999 konsorciem *W3C*. Syntax tohoto jazyka je opět založena na *XML*, a proto můžeme využít všechny dostupné nástroje, které dokáží pracovat s *XML*. V tuto chvíli existují tři verze. První byla vyvíjena v letech 1998 a 1999, přičemž veřejnosti byla představena v listopadu roku 1999. *XSLT 2.0* vyšla z neúspěšného pokusu vytvoření verze 1.1 v roce 2001. Proto se skupina *XSL* spojila s *XQuery* a v roce 2007 byla tato verze doporučena k užívání. [5] Mezi novinky patřily například regulární výrazy, více výstupů dokumentů nebo například funkce a operace pro práci s daty a časem. Poslední verzí je *XSLT 3.0*, kterou *W3C* přijalo v roce 2017. [6]

Jak již bylo naznačeno, *XSLT* je jazyk, který využíváme pro transformaci *XML*. Jeho hlavní funkcí je transformovat *XML* dokument do jiného *XML* dokumentu, *HTML* dokumentu nebo například textového souboru. Taktéž *XSLT* podporuje mezinárodnost, a tak se doporučuje využívání kódování *UTF-8*. [6]

Aby byla možnost získání konečné transformace, musíme využít obou dokumentů – *XML* a *XSLT*. Ty se následně transformují pomocí *XSLT* procesoru, ze kterého vyjde finální dokument. Větší firmy, jako *Oracle* či *Unicorn*, mají své vlastní procesory. Ale mezi nejznámější softwarové procesory patří *Saxon*, *libxslt/xsltproc* či *Xalan*. Microsoft má dokonce dva *XSLT* procesory – starší je známý pod jménem *MSXML* a novější, který je obsažen v *.NET*, se nachází v knihovně *Systen.Xml.Xsl*. [7]

1.2.2 **Dokument XSLT**

Celý dokument by měl být obsažen v párové značce „<xsl:stylesheet> </xsl:stylesheet>“. V této značce se následně zapíše verze a informace *xmlns*. Styly v *XSLT* se nazývají šablony, kdy každá část určuje, kde bude využitá a jak bude tato část transformována do konečného dokumentu. Šablona se defunuje pomocí „template“ a jmenného prostoru *xsl*. „<xsl:template match="x"></xsl:template>“. Samotný jazyk využívá *XPath* pro výběr určitých částí *XML* dokumentu, který je tvořen uzly. [8] Jeden takový dokument je možné vidět ve výpisu kódu č. 2.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ver
sion="1.0">
    <xsl:output method="xml" indent="yes"/>

    <xsl:template match="/Osoby">
        <root>
            <xsl:apply-templates select="Osoba"/>
        </root>
    </xsl:template>

    <xsl:template match="Osoba">
        <Jmeno ID="{@ID}">
            <xsl:value-of select="Jmeno" />
        </Jmeno>
    </xsl:template>
</xsl:stylesheet>
```

Výpis kódu č. 2: *XSLT dokumentu s návazností na XML dokument*

2 World Wide Web

World Wide Web je jednou ze součástí dnešního internetu. Jedná se o kolekci webových stránek, které jsou uloženy na serveru a ke koncovým uživatelům se dostávají právě prostřednictvím internetu. Z těchto tří slov dostáváme zkratku *WWW*, kterou zjednodušeně nazýváme jako *Web*. Začátek vývoje Webu se datuje k roku 1989, kdy Tim Berners-Lee se svými kolegy z organizace *CERN* začal pracovat na vymyšlení *WWW*. V roce 1990 tak vznikla první myšlenka zobrazení hypertextových (odkazových) dokumentů. Zavádějí se tak tři hlavní technologie – *HTML*, *URL* a *HTTP*. [9]

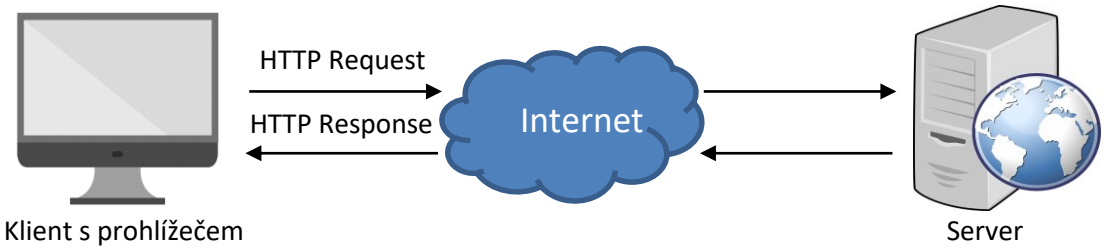
- *HTML – HyperText Markup Language* (Kapitola 3.2.1)
- *URL* – Vychází ze slov *Uniform Resource Locator*. Zjednodušeně se dá nazvat jako webová adresa, díky které máme přístup k dokumentům za využití internetu. [10]
- *HTTP* – Je zkratkou slov *HyperText Transfer Protocol*. Jedná se o aplikační protokol, jehož hlavním úkolem je starat se o komunikaci mezi klientem (většinou realizováno požadavkem prostřednictvím webového prohlížeče), a serverem. V moderní době se využívá protokol *HTTPS*, který je oproti originálu zabezpečený (*Secure*). [11]

Roku 1991 začínají fungovat prohlížeče, které byly ze začátku pouze v režimu řádků. Později toho roku ale vzniká grafické rozhraní, opět pod vedením Bernerse-Leeho. Pro vývoj *WWW* byl důležitý vznik prvního webového serveru mimo Evropu – v USA. Po proražení do Severní Ameriky tak začíná v roce 1993 práce na *X Window System* prohlížeči, který tak umožňuje vznik uživatelsky přívětivému prohlížeči, který je nazýván *Mosaic*, jenž je vytvořen organizací *NCSA*. *Mosaic* umožňuje větší rozsah v populaci, jelikož je možné ho využívat na jakékoliv platformě. Pro ilustraci – v roce 1991 existovalo 10 známých webových serverů, avšak na konci roku 1994 se toto číslo blíží k desetitisícům a celkových uživatelů webu je kolem 10 miliónů. [12]

I když *World Wide Web* zažíval rozmach a mohl by se stát formou vydělávání různých institucí, je stále prioritou využívat otevřené standardy (Open standards). Tohoto úkolu se zmocnil opět Berners-Lee, který tak v roce 1994 založil *International World Wide Web Consortium* (zkráceno na *W3C*). Hlavním důvodem vytvoření této společnosti bylo vedení webu k naplnění jeho celkového potenciálu vývojem společných protokolů, které mají za úkol podporovat celkový vývoj a interoperabilitu. [12] Ze začátku tak bylo potřebné najít světové partnery – těchto pozic se zmocnila v USA Massachusetts Institute of Technology, v Evropě se jednalo o European Research Consortium for Informatics and Mathematics (ERCIM) a v Asii tuto pozici převzala Keio University z Japonska. Momentálně má konsorcium *W3C* okolo 400 členů a vývoj webu nadále pokračuje. [13]

2.1 Fungování World Wide Webu

Základní funkce jsou prováděny na základě požadavků. V dnešní době se jedná o požadavek zaslaný prostřednictvím internetového prohlížeče na nějaký webový server, který na základě požadavku odešle odpověď. *WWW* tak funguje na principu klient–server. Server má tak za úkol shromažďovat informace o uživatelích nebo samotné soubory, které následně může transportovat díky svému webovému serveru. Klient je naopak ten, který o takové informace žádá, ale k přečtení těchto souborů potřebuje prohlížeč. Zjednodušeně se dá říct, že ve chvíli, kdy využíváme *URL* v adresním řádku nebo vyhledáváme cokoliv pomocí internetového vyhledávače, využíváme *WWW*. [12] Jednoduchý požadavek a odpověď je zobrazen na obrázku č. 1.



Obrázek č. 1: *Fungování World Wide Webu [14]*

2.2 WWW vs. Internet

Ačkoli se tyto pojmy často užívají ve stejném významu, je toto užití nesprávné a rozdíly mezi významy těchto pojmů jsou naprosto zásadní. Internet se považuje za síť, která propojuje zařízení (jako je například počítač, mobil, tablet). Pokud tak například posílám zprávy nebo chatuji, využívám internet. Na druhé straně máme *WWW*. *WWW* má za úkol zobrazování na základě požadavků webových adres. Klientovi je zobrazena vyrenderovaná webová stránka získána ze serveru. [15]

3 Webové stránky

Pod tímto pojmem se rozumí série několika dokumentů (existují i weby, které mají jeden jediný obsahový soubor a nazývají se single-page website. Veškeré informace jsou zde na jednom místě a odkazy jsou tvořeny pomocí křížků.), které jsou dílem jednoho autora nabízeným veřejnosti. Webové stránky tak obsahují samotné obsahové dokumenty, styly, skripty a multimédia. [16] Web se vždy zobrazuje ve webovém prohlížeči a jako první se uživateli po přístupu ukáže „domovská stránka," která se většinou prezentuje jako *index.html*. Pokud mají webové stránky více souborů, jsou vzájemně proklikatelné. [17]

3.1 Typy webových stránek

V oblasti webových stránek existují dvě nejznámější skupiny, a to statické a dynamické stránky. Obě skupiny mají své klady a zápory. Neexistuje všeobecná volba, který z typů je lepší. Vždy záleží na využití právě psané webové stránky.

3.1.1 Statické webové stránky

Jedná se o první využívanou skupinu. Ve zkratce se dá říct, že je to takový web, který nevyužívá dynamické generování obsahu. Jedná se tak o statické *HTML* soubory s multimédii, které jsou stylované pomocí kaskádových stylů, a možnosti jejich funkcionality nám zajišťují skripty psané zejména v *JavaScriptu*. Většinou se tak jedná o webové stránky, které jsou určeny pouze pro čtení.

V současnosti však statické stránky získávají nové funkce. Samotné stránky zůstávají statické, ale za využití vhodných pluginů jim můžeme přidat dynamické aspekty. Může se jednat o komentování prostřednictvím *Facebooku*, přidávání formulářů nebo vyhledávání pomocí *Google* a jejich *API*. Následně jen stačí napsat jednoduchý skript a využít to, co nám bude vráceno pomocí *API*. [18]

Mezi základní vlastnosti statických webů patří tyto:

- statické soubory jsou předávány rovnou uživateli
- žádné využití vlastností serverů (skripty spouštěny na serveru a databáze)
- bezpečnost

Soubory jsou předávány rovnou uživateli

Největší výhodou statických webů je to, že neexistuje žádná povinnost, aby byl takový web dostupný ve webovém prostoru. Jeho hlavním úkolem totiž je, aby každému jednotlivému uživateli v jakýkoliv časový okamžik, ukázal web v absolutně stejné podobě, jako byl vytvořen. Takový web tak můžeme předávat pomocí jakéhokoliv úložného zařízení (*CD* nosič nebo *USB* disk), jelikož se o jeho zobrazení stará prohlížeč a veškeré skripty jsou prováděny na straně uživatele. Web by ale měl být dostupný na internetu, o který se může starat hosting s webovým serverem (nejznámější je *Apache server* nebo *IIS server* od *Microsoftu*) a v tomto případě může být vhodný *FTP* protokol pro přenos souborů. V dnešní době je ale hojně využíván cloud. Takové služby dnes dokáží z jednoduchého úložiště dat vytvořit webový server. Není tak nutné se obávat o omezení velikosti webu a taktéž máme k dispozici jeho backup. [18]

Žádné využití serverových vlastností

Jelikož weby můžou být předávány jako klasické soubory a spouštěny na lokálních strojích, není možné využít například spouštění skriptů na serveru, jako může být *PHP*, nebo využívání různých databázových systémů. Samotná data jsou přístupná buď přímo v *HTML* souboru, nebo v nějakém jiném značkovacím jazyce, který data uchovává (například *XML*).

Bezpečnost

Jedná se o jednu z největších výhod oproti dynamickým webům. Jelikož zde není žádná databáze, neexistují žádné nesmyslné požadavky, které by ovlivnily funkčnost. Prohlížeč tak po serveru vyžaduje pouze *HTML*, *CSS*, *JavaScript* a multimediální soubory, a proto není možné, aby koncový uživatel udělal něco škodlivého vůči webu. Také neexistuje žádný důvod, aby někdo takové soubory napadal. O samotnou bezpečnost se sice stará webový server, ale pokud už k útoku dojde a server se s ním nedokáže vypořádat, je snadné celý statický web nahrát na server znovu a nepřijít o žádná data. [19]

Mezi největší výhody patří rychlost a právě bezpečnost. V současnosti je rychlost webu jednou z nejvýznamnějších vlastností. Vzhledem ke statickým *HTML* souborům, které se zobrazují každému jednotlivému uživateli stejně, a také nevyužívání žádných dat z databáze a ani žádných skriptů na straně serveru máme jistotu, že web nebude ničím zpomalován a bude rychlý i díky snadnému cachování statických souborů. V otázce bezpečnosti nám tak odpadají starosti ohledně možných útoků. V dnešní době bývají využívané útoky zejména v podobě *cross-site scripting* a vzhledem k využívání databází také *SQL injection*. Pokud se budeme nadále držet bezpečnosti, je také vhodné říct, že ani využívání samotných *CMS* systémů není bezpečné. Hackeři stále dokáží využívat jejich zranitelnosti a mezer v kódu.

Jako další z výhod se dá zmínit také verzování či flexibilita. Verzování nám umožní vrácení ke starým verzím webu, pokud nastane chyba, neexistuje nic snazšího než si nahrát poslední stabilní verzi. U flexibility se dá říci, že to, jak samotný web bude vypadat a fungovat, záleží jen a pouze na nás, neexistují žádné limity. [18]

Nejvhodnější druhy statických webových stránek

Statické weby nejsou vhodné pro všechny. Určité druhy stránek vyžadují dynamická data po zavolání různých *API* či využití služeb třetí strany, i když i tato vlastnost začíná být více než možná také na statických webech. Statické webové stránky je tak vhodné využívat v takovém případě, kdy víme, že na stránkách nepotřebujeme dynamicky personalizovaná data nebo samotný obsah. Další z důležitých předpokladů je četnost aktualizace obsahu. U statických webů tak jedna změna v jednom *HTML* dokumentu nezajistí aktualizaci v jiném *HTML* dokumentu. Z tohoto důvodu je nutné provádět mnoho kopírování a oprav v samotném kódu.

Pokud víme, že nebudeme potřebovat velké množství uživatelských interakcí a personalizaci zobrazovaného obsahu, tak se mezi nejvhodnější typy takových webů řadí: [18]

- Blogy
- Dokumentace
- Informační stránky

Blogy

Blogy se dají považovat za nejvyužívanější typy statických webů, a proto mnoho generátorů nabízí vhodné šablony s přívětivými styly. Jelikož jsou blogy zejména zaměřené na předávaný obsah a interakce s uživatelem se omezuje pouze na komentování, tak se jedná se o vhodnou variantu statických webových stránek. [18]

Dokumentace

Další z velmi využívaných typů jsou dokumentace. Jejich výhodou je to, že se jedná o čistě statické stránky se stálým obsahem. Samotná aktualizace obsahu není nijak častá a ani náročná. Interakce s uživatelem se omezuje pouze na zobrazení stránky. Samotný hosting tak není náročný a tuhle funkci může zastat například *GitHub*, už jen díky svému udržování minulých verzí. [18]

Informační stránky

Jedná se o poslední z nejvíce doporučených typů. Jejich funkcí je koncového uživatele pouze informovat o službě, kapele nebo například události. Jedná se tak o nejjednodušší formu webu, kdy je sice důležité udržovat web aktualizovaný, ale při využití statických generátorů se jedná o velmi přívětivou variantu. [18]

3.1.2 Dynamický web

Druhou variantou je dynamický web. Mezi jeho největší výhodu se řadí to, že obsah, který je zobrazen uživateli, je vždy generován v reálném čase. Stránky jsou tak uživateli předány přesně na míru. Často se jedná o jazykové provedení, časovou zónu a podobně. Může se tak stát, že stejný web u dvou různých uživatelů nebude mít stejnou podobu. Abychom této vlastnosti mohli využít, musí být web umístěn na serveru a využívat jeho vlastností. Většinou tak využívá pro ukládání dat databází a skriptovacích vlastností. Existují celkem dvě možnosti – obsah tvořen na straně uživatele a obsah tvořen na straně serveru. [20]

Obsah tvořen na straně uživatele

Jedná se o takovou variantu, kdy je finální obsah či funkce stránky závislá na akcích uživatele. Může se jednat o stisknutí klávesy či využití jakékoliv funkce myši. Prohlížeč si prvně ze serveru získá samotné webové stránky a až poté jsou prováděny kódy skriptů, které vykonává prohlížeč a aktualizuje tak obsah webu. V takovém případě se standardem stal *JavaScript*, hojně oblíbený byl také *Flash*. [21] Současně je tato vlastnost využívána i v případě statického webu v menších rozměrech.

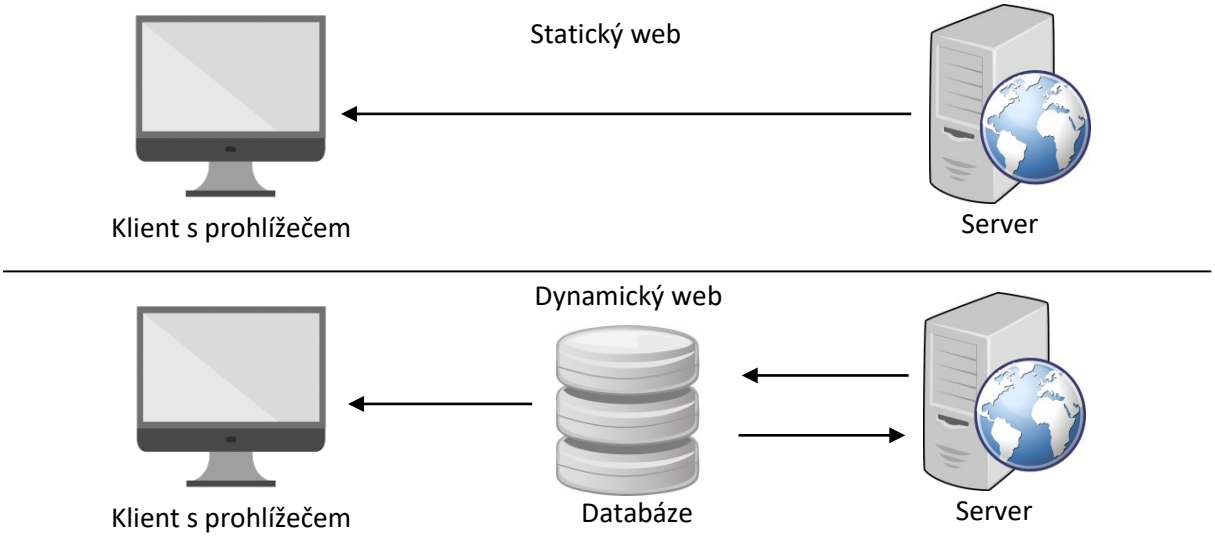
Obsah tvořen na straně serveru

Taková webová stránka, která se změní ve chvíli, kdy je načtená či navštívená, využívá *server-side scripting*. Tento obsah ze strany serveru se vygeneruje, když je web načten. Jako příklad se dá využít e-shop a jejich nákupní košík nebo přihlašovací stránky. V tomto případě jsou obvyklé jazyky jako *PHP*, *JSP* nebo například *ASP.NET*. Neexistuje proto žádný standard. Vždy záleží na tom, jaký server je využíván. [20]

Využití dynamického webu je vhodné ve chvíli, kdy víme, že náš celkový web bude mít stovky až tisíce podstránek. Nemusíme se starat o aktualizaci každé stránky a stačí se pouze postarat o aktualizaci určitého řádku v databázi. Pokud tak chceme unikátní obsah pro každého uživatele, nevadí nám obsah uložen v databázi a nevadí nám strach z napadnutí, jsou dynamické stránky vhodnou volbou.

3.1.3 Statický vs. dynamický web

Nejdůležitějším krokem ve výběru druhu webu je zjištění, k čemu má náš web sloužit a jaké jsou naše všeobecné znalosti. Pokud nám nevadí, že web bude mít stálou statickou podobu, ale máme nad ním absolutní kontrolu s největším zabezpečením, je statický web vhodný. Stačí nám znát základní znalosti ohledně *HTML* a *CSS*, abychom vytvořili přívětivý web. Pokud však považujeme za nutné mýti dynamický obsah, musíme sáhnout po dynamickém webu. Takový druh je vhodný zejména například pro e-shop. K otázce znalostí nám stále stačí stále *HTML* s *CSS*, ale už přibývá nutnost znalosti skriptovacích jazyků, aby se náš web stal skutečně dynamickým. Větším negativem se ale jeví bezpečnost. U statických webů máme vše pod kontrolou a napadnutí webů je takřka neopodstatněné, u dynamických webů je ale napadnutí samotných databázi snadné, také je velmi snadné využití mezer v samotných skriptech. V konečném důsledku ale záleží čistě na nás, do jakého typu webu se chceme pustit. Na obrázku č. 2 můžeme vidět rozdíl mezi využívanými fyzickými komponenty statického a dynamického webu. [22]



Obrázek č. 2: *Statický vs. dynamický web [14]*

3.2 Složení statického webu

Abychom získali finální podobu webové stránky, musí obsahovat minimálně jeden dokument – *HTML* soubor. Pokud bychom chtěli tomuto *HTML* obsahu přidat vzhled, využijeme *CSS* souboru. A jestliže má mít web nějakou vlastní funkcionalitu, využijeme skriptů – *JavaScript*.

3.2.1 HTML

HTML je zkratkou pro *Hypertext Markup Language*. Tento jazyk je značkovací a využívá se pro tvorbu webových stránek. Za jeho počátek se považuje rozmezí mezi lety 1990 až 1995, kdy probíhalo několik revizí a rozšíření, o které se staralo *CERN* a *IETF*. Na tento jazyk měl vliv vznik *W3C*, kdy tak postupně začala vznikat verze 3.0, která byla ale neúspěšná, ale umožnila cestu k verzi 3.2, která při vývoji využila pragmatičtější cesty. Po nástupu verze 4.0 ale začíná *W3C* zastavovat vývoj jazyka *HTML* a podporuje vznik *XML* verze *HTML – XHTML*. Po neúspěšném pokusu prosazování *XHTML* se rozhodlo o pokračování vývoje v roce 2011 a vzniká tak poslední verze 5.0, která opravuje veškeré chyby a nedostatky verze předchozí. [23]

HTML dokument musí obsahovat několik značek:

- „<!DOCTYPE html>“ - tato značka definuje typ dokumentu, který dostane prohlížeč.
- „<html></html>“ - jelikož se jedná o značkovací jazyk, musíme tak zajistit kořenový prvek, který obaluje veškerý obsah.
- „<head></head>“ - v hlavičce se obvykle nachází titulek dokumentu „<title>Titulek</title>“ a kódování dokumentu v *meta* značce, kde se nastavuje atribut *charset*. V tomto bloku se mohou nacházet cesty k *CSS* stylům, ikoně či využívanému fontu z internetu v párové značce „<link>“. Také se zde může objevit párová značka „<script>“, která definuje zejména funkcionalitu za využití *JavaScriptu*.
- „<body></body>“ - jedná se o poslední značky, ve kterých se nachází celkový obsah dokumentu. Můžeme zde tak vidět značky jako „<h1></h1>“, které definují nadpis, „ “ jsou určeny pro odstavce a nepárová značka „
,“ která vynechá jeden řádek.

Takový soubor je možné vidět na výpisu č. 3.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Titulek</title>
    </head>
    <body>
        <h1>Nadpis první úrovně</h1>
        <p>Ukázka odstavce a odkazu <a href="#">odkaz</a>.</p>
        <!-- Ukázka komentáře. -->
    </body>
</html>
```

Výpis kódu č. 3: *Ukázka HTML dokumentu*

3.2.2 CSS

Cascading Style Sheets neboli *Tabulky kaskádových stylů* mají za úkol přiřadit k dokumentu *HTML*, *XHTML* nebo *XML* jeho grafické vykreslení. Z počátku se počítalo, že tento úkol převezme *HTML* dokument, dokonce existovaly zavedené značky, jako jsou „<font>“ nebo „<big>“ s atributy *bgcolor* či *align*. Avšak v závislosti konkurenčních bojů prohlížečů se od této cesty upustilo a standardizační organizace světa webů *W3C* začíná více rozvíjet a prosazovat cestu *CSS* stylů a zajišťuje nám tak separaci obsahu a vzhledu webu. K dnešnímu dni existují celkem tři verze *CSS1 – CSS3*. [24] Syntax jazyka je velmi snadná. Vždy je potřeba vybrat, o jaký selektor se jedná, a k němu do složených závorek vybrat vlastnost a nastavit její hodnoty (nemusí být jen jedna) zakončené středníkem. Jeden takový kaskádový styl je k vidění ve Výpisu kódu č. 4.

```
body{
    color:red;
}
/*
Na různe třídy v HTML dokumentu odkazujeme pomocí tečky
U identifikátoru to je hash.
*/
.trida, #idt{
    font-size: medium;
}
a:hover{
    text-decoration: none;
}
```

Výpis kódu č. 4: *Ukázka CSS dokumentu*

3.2.3 JavaScript

JavaScript je objektově orientovaným skriptovacím jazykem. Jeho začátky se datují do roku 1995. Původně se jednalo o způsob, jak do webových stránek přidat programy v prohlížeči *Netscape Navigator 2.0*. Za zmínku stojí, že slovo *Java* v názvu nemá nic společného s tímto programovacím jazykem, a jedná se o pouhý marketingový tah k získání pozornosti. [25] Z počátku existovala verze *LiveScript* (následně označována jako *JavaScript*), implementována pro *Netscape Navigator*. Avšak *Microsoft* nechtěl nechat nic náhodě, a tak vytvořil svůj *JScript* pro *Internet Explorer*. Oba tyto skriptovací jazyky se v lecčem odlišují a v konečném důsledku existuje x různých variant pro různé verze internetových prohlížečů, což vedlo v roce 1997 k přání vývojářů po standardizaci *JavaScriptu*. Standardizační orgán *ECMA* tak zavádí *ECMAScript*, který se stává označením standartu *JavaScript*. [26] Největší výhodou tohoto jazyka je to, že samotný skript je spouštěn na straně uživatele, a tak není potřeba mít webovou stránku nahranou na nějakém serveru. Máme tak možnost vytvářet různé animace či využívat interaktivní prvky.

```
//Ukázka deklarace proměnné a následného výběru prvku v HTML dokumen
tu a zapsání textu
function ukazka() {
    var text = "Hello World!";
    document.getElementById("idnt").innerHTML = text;
}
```

```
/*
Ukázka rekurzní funkce na bázi výpočtu
faktoriálu a návratové hodnoty
*/
function faktorial(n) {
    if (n === 0)
        return 1; // 0! = 1

    return n * factorial(n - 1);
}
```

Výpis kódu č. 5: *Ukázka JavaScript souboru*

4 Tvorba webu

Existují dva typy tvorby webového obsahu. Avšak abychom mohli takový web vytvořit, můžeme využít různých generátorů. Opět existují dva – dynamický a statický. Weby ale není vždy nutné vytvářet za využití generátorů. Existuje také možnost absolutní svobody – napsání samotné struktury *HTML* dokumentu s obsahem, vytvoření různých skriptů a zejména jejich stylů. Tato volba se ale může zdát časově náročná a někdy i nepříjemná.

4.1 Generování dynamického webu

První z možností je generování dynamických webů. Jako základ se využívá zejména jazyku *PHP*. V tomhle případě není potřeba psát program pro generování *HTML* dokumentů. Jednoduše se mezi *HTML* značky přidají skripty psané v *PHP*. Abychom ale zajistili větší dynamičnost, může se využít také databázového systému *MySQL*, který uchovává ve svých tabulkách veškerý obsah webu. [21]

4.1.1 CMS – Content Management System

Jedná se o systém, který využívá *PHP* a databázových systémů. Pro správu webových stránek se využívá jeho webového rozhraní, kdy máme jeho prostřednictvím přístup k editaci. Výhodou *CMS* je to, že obsah je uložen na serveru, není také nutná jakákoliv znalost programovacího jazyka a taktéž přístup je možný odkudkoliv. Mezi nejznámější systémy pro správu systémů patří *Wordpress* a *Drupal*. [27]

Drupal

První verzi *Drupalu* vytvořil Dries Buytaert v roce 2000. Hlavní myšlenkou bylo sdílení informací a novinek kolegům z univerzity, avšak bez využití internetu. Za oficiální první verzi *Drupalu* se však považuje ta, která byla vytvořena v roce 2001 s plnou podporou online prostoru.

Samotný *Drupal* je open source a jeho největší boom nastává v roce 2011, kdy se stává uživatelsky přívětivý, zdokonaluje zabezpečení a zavádí multijazyčnost. Tento *CMS* je vhodný zejména pro tvorbu propracovanějších projektů s potřebou různých vlastních typů obsahu. [28]

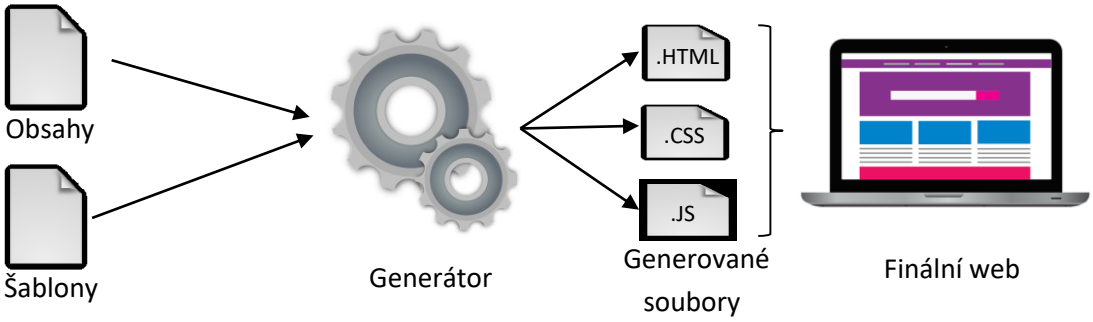
WordPress

Historie *WordPressu* se datuje již do roku 2001, kdy se skupinka nadšenců snažila vytvořit logicky strukturovanou, snadno upravitelnou a zejména rozšiřitelnou platformu. V roce 2003 se *WordPress* představil jako čistě blogovací platforma, avšak postupně se vývoj dostával do pozice plnohodnotného *CMS* systému. V dnešní době patří *WordPress* k nejvyužívanějším systémům s celkovým podílem 35, 2 %. Tento systém je vhodný zejména pro osoby, které nejsou tak znalé v oblasti IT, jedná se o velmi jednoduchou platformu a oproti 18 tisícům modulů, které nabízí *Drupal*, má *WordPress* v nabídce více, jak 21 tisíc. [29]

4.2 Statické generátory webu

Jedná se o další možnost tvorby webu. V tomto případě se jedná o vytvoření statických stránek s tím rozdílem, že psaní *HTML* dokumentů, *CSS* stylů a *JavaScript* skriptů není na nás. Hlavním

smyslem statických generátorů je to, že generátoru předáme veškerá dynamická data a obsah, ze kterých se vytvoří statický web.



Obrázek č. 3*: Ukázka generování statického webu [14]*

V dnešní době existují stovky generátorů, které jsou psány v různých jazycích a využívají jiných vzhledových šablon, ale v základu se snaží udržet většinu hlavních vlastností, kterými jsou: [30]

- psaní v jednom programovacím jazyce
- několik template jazyků pro vytvoření vzhledů
- využití příkazových řádků k vytváření webu
- data jsou předávána v textovém souboru
- statický generátor má vestavěný server
- možnost rozšíření

Jednotný programovací jazyk

Každý z generátorů je vytvářen v jednom jazyce. Z praktického hlediska by totiž nedávalo smysl, aby se o každou část webu staral jiný jazyk či úplně jiný program, který není nijak kompatibilní s dalším jazykem. Z tohoto důvodu tak existuje na trhu větší množství generátorů, přičemž je každý generátor psán v jednom zvoleném jazyce a my si tak vybíráme, který jazyk je nám nejbližší. [30]

Vzhled webové stránky

Pokud je samotný generátor psán v jednom jazyce, není tomu tak u vytváření vzhledu stránky. Samozřejmě můžeme nadále využívat jen jedné šablony a té vše přizpůsobit, ale také existuje varianta propojování externího jazyka a *HTML* tagů. Stránka je tak následně generována na základě různých podmínek či cyklů, které zapíšeme externím jazykem.

Samotný vzhled je v tomto případě vytvářen za použití specifických šablon, které se ve většině případů nejvíce hodí k vybranému programovacímu jazyku. [30]

Příkazový řádek

Pro veškerou práci s generátorem se využívá příkazového řádku. V defaultní podobě tak nejsou generátory vyvíjeny s grafickým rozhraním pro interakci. Pro práci s generátory je tak nezbytná znalost základních příkazů, které jsou ale lehce dohledatelné v dokumentaci. [30] Vývojáři statických generátorů však nezahálejí a začínají přidávat nástroje pro nadstavbu takového rozhraní. Většinou takový systém funguje na principu využití *GitHubu* či *Dropboxu*, který obsahuje tento web, a veškeré další úpravy jsou prováděny v prohlížeči. [18]

Předávání dat

V porovnání s manuálním vytvářením statického webu se v tomto případě nevyužívá zapisování obsahu stránek rovnou do určitých *HTML* souborů. Generátor k převodu do *HTML* potřebuje textové soubory se správným názvem stránky. Jako soubory se tak využívají například *JSON*, *YAML* nebo *Markdown*. Generátor tak má za úkol provést konverzi tohoto souboru do finálního *HTML* souboru, který je zobrazen uživateli. [30]

Vestavěný server

Další důležitou vlastností je vestavěný server. Jeho funkce je vhodná ve chvíli, kdy nás zajímá aktuální podoba generovaného webu. Není tak potřeba pokaždé vygenerovat novou webovou stránku, ale až následně zhodnotit její výslednou podobu a provést potřebné změny. Tato varianta je velmi praktická a celkově ubírá na náročnosti samotného generování webu. [30]

Rozšiřitelnost

Poslední důležitou součástí je přidávání externích pluginů do našich webů. Ve většině případů existují předpřipravená rozšíření a nám tak stačí ze seznamu vytvořených pouze zvolit ten, který se nám nejvíce hodí. Pokud jsme však v situaci, že plugin neexistuje, většina generátorů umožňuje vytvoření čistě vlastních. [30]

K využití statických generátorů jsou potřebné určité znalosti a návyky. Ze začátku je jednoznačně nejdůležitější umět pracovat s *HTML*/*CSS*/*JS*, jako by se jednalo o normální tvorbu webu. Mezi nadstavbu se pak řadí tyto body, které vycházejí z vlastností generátorů:

A) Komfortní práce s příkazovým řádkem

V dnešní době je mnoho vývojářů zvyklých na práci s vývojovým prostředím, které se stará o většinu pomocných prací (například slovník s návrhem funkcí či datových typů). Taková varianta ale zde není standardem. Je tak nutné se s příkazovým řádkem podrobněji seznámit a využít všech jeho silných stránek. Dnes mají populární SSG různé nadstavby pro vyhnutí se příkazovému řádku, ale tato varianta stále zůstává standardem a také formou zpětné vazby především pro debuggování a vypisování chyb.

B) Schopnost se učit a pracovat s doplňkovými jazyky a nástroji

Jak již víme, SSG vyžadují pro tvorbu obsahu ukládání dat v textových souborech. Na nás tak zůstává, abychom se vypořádali se specifiky určitých jazyků. Jelikož každý generátor využívá různých jazyků a jejich kombinací, nebude tak stačit umět pouze *Markdown*, ale budeme se muset naučit pracovat i například s *JSONem*, *Liquidem*, *YAMLem* a mnoha dalších.

C) SSG jsou pro vývojáře

Mnoho vývojářů v dnešní době obvykle pracuje bez jakéhokoliv grafického rozhraní, avšak pro laiky se tato forma může zdát nadmíru komplikovaná, ve které je snadné se ztratit. [18]

4.2.1 Volba statického generátoru

O každém generátoru si můžeme pomocí vyhledávače či jejich vlastních stránek zjistit jeho kvalitu. Důležité je si zvolit takový generátor, který je psán v jazyce, jenž je nám blízký a u kterého známe jeho klady a zápory. Dalším důležitým prvkem je dokumentace. Většinou pracujeme s novými

věcmi, které zatím neznáme, a právě přehledná a aktualizovaná dokumentace nám pomůže k lepší orientaci. Posledním a pravděpodobně nejdůležitějším bodem je aktualizovanost vybraného generátoru. Pokud se na jejím vývoji stále pracuje, je výběr správný, pokud však SSG nemá několik měsíců či let žádné aktualizace, je důležité se takové variantě vyhnout. Může totiž nastat situace, kdy má generátor několik nedostatků, ale neexistuje žádná aktualizace, která by vedla k řešení. Může se také stát, že narazíme na vlastní problém, na který zatím nebyla žádná zpětná vazba, a tak se musíme spoléhat pouze na vlastní řešení. [18]

4.2.2 Nejvyužívanější generátory statického webu

V dnešní době existují stovky generátorů. Většina z nich je v databázi na stránce www.staticgen.com/, kde existují žebříčky oblíbenosti. Na předních příčkách nalezneme *Jekyll*, *Hugo*, *Next.js* a *Pelican*, o kterých je zde společně s generátorem *Hyde* podrobněji pojednáváno. Každý z těchto generátorů se vyznačuje jinými programovacími jazyky. Nelze však přímo říci, který z těchto generátorů je lepší volbou, vždy záleží na preferenci uživatele. Nejdůležitější je však komfortní znalost jazyka. V praktické části této bakalářské práci je užíváno generátoru *Pelican*, a proto je v této kapitole výrazněji rozebírán.

Jekyll

Tento generátor je psaný v jazyce *Ruby* a jako šablonu vzhledů využívá *Liquid*. Mezi pozitivní stránky *Jekyllu* se řadí zejména to, že obsah můžeme předat buď jako soubor *Markdown* nebo můžeme využít *HTML* souborů. Také využívá datové soubory *YAML*, které jsou přiřazeny pro navigační menu. *Jekyll* je tak vhodný pro normální webové stránky nebo blogy. [31]

Hugo

Naopak *Hugo* je psaný pomocí jazyka *Go*, který je také využíván k šablonám vzhledů. *Hugo* se vyznačuje svou rychlostí, a také několika možnostmi pro předání obsahu – buď se může jednat o známý *Markdown*, *reStructuredText*, *HTML*, nebo například *MMark*. [32]

Next.js

Předposlední z vybraných generátorů je ten, který je psán v jazyce *JavaScript*, a pro své šablony využívá *React*. Tento generátor je postavený na .js, .ts nebo .tsx souborech. Tyto soubory jsou tak složeny z funkcí, které vrací *HTML* značky s textem. *Next.js* se dá sice považovat za statický generátor, ale umožňuje také generaci dynamického obsahu. [33]

Hyde

Hyde je statický generátor psaný v jazyce *Python* a *Jinja2*. V určitou chvíli býval udržovaný a aktualizovaný i pro další verze *Pythonu*, avšak s příchodem *Pythonu 3* se jeho vývoj zamrazil s poslední verzí vydanou 10. 3. 2016. Tento generátor má několik příjemných vlastností, jako je například konfigurovatelné řazení, tagování a zařazování do skupin. [34]

4.2.3 Pelican

Pro potřeby této bakalářské práce je větší pozornost věnována generátoru *Pelican*. Jedná se generátor statických webových stránek, který nevyužívá provádění skriptů na straně serveru a také nemá vlastní databázi. Jeho struktura je psána v jazyce *Python* a k tvorbě vzhledů se tedy využívá *Jinja2*.

Tento generátor je vhodný pro vytváření blogů nebo klasických webových stránek. Jak již bylo řečeno, i tento generátor umožňuje možnost rozšíření. Může se tak jednat o pluginy, které jsou předem dostupné, anebo existuje možnost vytvoření vlastního pluginu, který je pak následně využit na webu. Další možností rozšíření je vlastní vzhled, který ale musí být psán v jazyce Jinja2, a to pouze z důvodu udržení vlastností tohoto generátoru. Samotný obsah je tvořen pod značkovacím jazykem *reStructuredText* nebo existuje možnost využití *Markdown*, avšak preferovanější je první zmíněný. *Pelican* nám také umožňuje upravení nastavení, jako je například přidání vybraných pluginů a jejich cest nebo nastavení statických cest. Další výhodou je to, že *Pelican* umožňuje využití externích doplňků, jako je například *Twitter*, *Google Analytics* a další. Také nejsme omezeni na jeden jazyk, neboť obsah může být publikován v jakémkoliv jazyce. [35]

Markdown

Pro práci s SSG *Pelican* budeme využívat jako jazyk pro tvorbu obsahu *Markdown*. *reStructuredText* je sice hojně podporovaný Pythonem, ale jednoduchost, síla, přehlednost a velká využívanost nás přesvědčila pro *Markdown*.

Jedná se o odlehčený značkovací jazyk, který umožňuje přidávání formátovacích značek do textových dokumentů a také převedení do dokumentu, který je publikovatelný na webu (zejména *HTML*, ale může se jednat i o zastaralejší *XHTML*). O tento jazyk se zasloužil John Gruber v roce 2004, kdy mu jako inspirace sloužil prostý text e-mailu. [36]

V tomto jazyce existuje několik typů značek, přičemž žádná nevyužívá písmen. Nadpisy vždy využívají znaku „#“ a počet jeho opakování určí úroveň nadpisu, jako by se jednalo o *HTML*. „##Nadpis“ tak bude v *HTML* „<h2>Nadpis</h2>“. Pro psaní odstavců se nevyužívají žádné značky, text tak můžeme psát plynule za sebou. Ve chvíli, kdy chceme ukončit řádek, napíší se na konec dvě mezery. Ukončení odstavce se provede enterem. Pro zvýraznění textu se využívá značek „*“ a „_“. Jestliže chceme využít kurzívu, napíše se tato značka jednou „_kurzíva_“, kdy se místo podtržítka může využít hvězdička. U tučného písma se tyto značky napíši dvakrát „__tučně__“ a opět lze podtržítka nahradit opět hvězdičkami. Mezi další často využívané značky se řadí odrážky a číslování. Pokud chceme využít odrážky, píšeme text za pomlčku s mezerou. Pro více úrovní odrážek se využívá tabulátoru. U číslování se využije stejného způsobu, akorát pomlčku nahrazuje „1.“. Pro vytváření obsahu jsou důležité obrázky či odkazy. Pro napsání odkazu se využívá hranatých závorek s textem, který je zobrazen uživateli a za ním se nachází cesta – „[Text](http://www.vsb.cz/)“. U obrázků se před hranaté závorky přidá vykřičník. Pokud se v obsahu nachází tabulka, tak pro oddělení hlavičky tabulky využijeme minimálně tři pomlčky, pro oddělení sloupců se využívá svislá čára. Mezi poslední nejvyužívanější prvky se řadí psaní kódu. Stačí nám tak vzít jakýkoliv kód, vložit ho mezi značky „```“ a na začátek jen napsat, o jaký programovací jazyk se jedná.

Markdown je v dnešní době velice užívaný v mnohých statických generátorech, jedná se o přehledný jazyk, rychlý a srozumitelný. [37]

Struktura generátoru

Pelican má za úkol vzít několik seznamů dokumentů a vytvořit z nich výstupní dokumenty. Jak bylo řečeno, můžeme využít *reStructuredText* nebo *Markdown*. Avšak není to vždy povinnost, v konečném důsledku můžeme zvolit jako vstup a výstup jiné formáty. Je nutné ale upravit či vytvořit tři typy: Jedná se o *reader*, *writer* a také *generator*. *Reader* má za úkol přijímat značkovací soubory

s daty. Následně tak stačí vytvořit novou funkci v *Pythonu* pro tento úkol, která bude vracet *HTML* soubor. Druhý typ je *writer*, který zodpovídá za vytváření finálních dokumentů, jako jsou *HTML* soubory nebo *RSS feed*. Posledním typem je *generator*. Ten obsahuje dvě funkce – *generate_context* a *generate_output*, ale nejsme je povinni definovat obě. *Generate_context* je volán první a většinou vyhledává všechny možné stránky, které převede do objektů a vyplní jimi kontext. Následně je volán *generate_output*, který jen převezme tento kontext a pomocí *writeru* vytvoří konečný výstup. Samotný *Pelican* je otevřen novým věcem k učení a je jen na nás, zda využijeme předem doporučený formát nebo se pustíme do něčeho nového. [35]

Vzhled webové stránky

Právě vytváření vzhledů je prostorem ke zkoušení nových věcí. Existují tedy jak předem implementované vzhledy v *Pelicanu*, dále také vzhledy vytvořené komunitou a publikované na *Git*H*ubu* nebo lze opět vytvořit něco vlastního. Máme však jednu jedinou podmínku, a to psát tento vzhled pomocí jazyku *Jinja2*. *Jinja2* je jazyk pro vytváření šablon vycházející z *Django*. Většinou se tak jedná o šablonu oddělenou od kódu. Nahrazují se buď klíčová slova zapsána ve složených závorkách „{{nahradit}}“ nebo se vykoná kód, který je psán v bloku „{%kod%}“. Proměnné, které v těchto šablonách budeme využívat, musí mít stejný název jako ty, které jim budeme předávat prostřednictvím nastavovacího souboru *pelicanconf.py* nebo značky v souboru *Markdown*. [38] *Pelicanu* pak jen stačí při vytváření obsahu říct, kde tento styl nalezne za značkou -t: „-t /projekty/web/themes/muj_vzhled“.

Využívané výhody Pelicanu

Dalšími z důvodů, proč se v této práci užívá *Pelican*, jsou šikovná rozšíření a editační vylepšení. Jedná se o definování vlastní chybové stránky kódu 404. Nejsme tak nucení využívat defaultní podobu a vytvoříme svůj vlastní 404.md (*Markdown*) soubor. Další z vychytávek je publikace na *Git*H*ub*. V dnešní době je preferováno své dílo zálohovat nebo se o něj podělit s dalšími uživateli. *Pelican* tak umožňuje na *Git*H*ub* dvě verze publikace – User Pages a také Project Pages. Veškerá tato činnost zůstává v rozhraní příkazového řádku pomocí dvou základních příkazů „ghp-import output -b gh-pages“ následován „git push“, kdy u User Pages se dopíše cesta, kam se budou tyto soubory nahrávat „git@github.com:elemoine/elemoine.github.io.git gh-pages:master“, při využití Project Pages se zapíše jen tradiční origin „origin gh-pages“. Pelican také umožňuje importování již existující stránky a může se jednat o tyto: *Tumblr API*, *WordPress XML* export nebo například *RSS/Atom feed*. *Pelican* je tak vhodný ve chvíli, kdy chceme opustit „dynamický svět“. Není tak nutné začínat od nuly. [35]

4.2.4 Porovnání generátorů

V tabulce č. 1 můžeme vidět přehledné porovnání všech zmíněných generátorů.

Tabulka č. 1: *Porovnání využívaných statických generátorů*

	Jekyll	**Hugo**	**Next.js**	**Pelican**
Programátorský jazyk	Ruby	Go	JavaScript	Python
Template jazyk	Liquid	Go	React	Jinja2
Obsah	HTML, Markdown	Markdown	JavaScript, TypeScript	reStructuredText, Markdown
Poslední větší aktualizace	20.8.2019	20.2.2020	15.1.2020	17.10.2019
Počet témat	>= 380[1]	>= 201[2]	>= 28[3]	>= 127[4]
Počet pluginů	>=52[5]	>=24[6]	>=27[7]	>=49[8]
Možnost importu	>=24[9]	Jekyll, Ghost, Octopress, DokuWiki,WordPress,Medium, Tumblr, Drupal, Joomla, Blogger, BlogMl, Contentful[10]	Ne	Blogger, Dotclear, Posterous, Tumblr, WordPress, RSS/Atom[11]
Rychlost generování	Pomalejší, ale možnost zrychlení[12]	Velmi rychlý[13]	Pomalejší[14]	Pomalejší
Grafická editace	Ano (Jekyll Admin, Prose)[15]	Ano (Netify CMS, enwrite, Hokus CMS)[16]	Ano (Netify CMS, Strapi, Cockpit a další)[17]	Ano (shoebill[18], Pelican-GUI[19])
Možnosti nasazení	Github, Rsync, Amazon S3, FTP[20]	Wercker, Rsync, Github, využití STFP skrz Tools[21]	ZEIT Now, GitHub[22]	Rsync, GitHub[23]

[1] http://jekyllthemes.org/
[2] https://themes.gohugo.io/
[3] https://themeforest.net/search/nextjs/
[4] https://forestry.io/blog/hugo-vs-jekyll-benchmark/
[5] https://www.reddit.com/r/reactjs/comments/ap3d0h/nextjs_slow_to_build/
[6] https://gohugo.io/tools/
[7] https://github.com/zeit/next-plugins
[8] http://shortcircuits.io/pelican-plugins-site/
[9] https://import.jekyllrb.com/docs/home/
[10] https://gohugo.io/tools/migrations/
[11] https://docs.getpelican.com/en/stable/importer.html
[12] https://forestry.io/blog/hugo-vs-jekyll-benchmark/
[13] https://forestry.io/blog/hugo-vs-jekyll-benchmark/
[14] http://www.jekyll-plugins.com/plugins
[15] https://headlesscms.org/projects/jekyll-admin/
[16] https://gohugo.io/tools/frontends/
[17] https://headlesscms.org/
[18] https://github.com/FedericoCeratto/shoebill
[19] https://github.com/khrogos/pelican-gui
[20] https://jekyllrb.com/docs/deployment/manual/
[21] https://gohugo.io/hosting-and-deployment/
[22] https://nextjs.org/docs/deployment/
[23] http://docs.getpelican.com/en/3.6.3/publish.html

5 Generování statických webových stránek

5.1 Stránky za využití Pelicanu

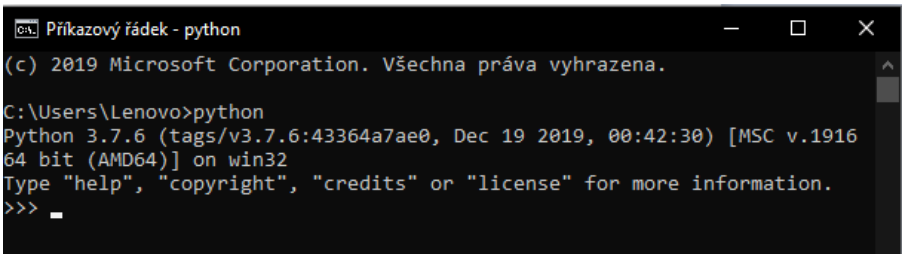
V této bakalářské práci užíváme statický generátor *Pelican*. Jak již bylo zmíněno výše, jedná se o takový SSG, který je postaven na jazyce *Python*. V naší práci využíváme verzi *Pythonu 3.7.6*, která je kompatibilní s verzí *Pelicanu 4.2.0*. Pro veškerou práci je využito operačního systému *Microsoft Windows 10*.

Pro vytváření takového webu využíváme informační stránky Petra Šalouna, které sloužily k bývalému kurzu *Programování I*, jež jsou dostupné na této webové adrese – http://www2.cs.vsb.cz/saloun-zp/doku.php?id=start. Pokusíme se tak přepsat tento web do statické podoby, ve které by bylo vhodné udržet základní strukturu celých stránek, jejich propojení a také využití vyhledávacího prvku.

5.1.1 Instalace Pythonu

Abychom mohli *Pelican* využít, musíme mít na svém zařízení nainstalován *Python*. K tomu nám poslouží oficiální stránky https://www.python.org/downloads/release/python-376/. Následně pak doporučujeme vybrat „Windows x86-64 executable installer“ a provést tradiční instalaci. V této práci je provedena standardizovaná instalace s původním nastavením, avšak s přidáním „Add Python 3.7 PATH“ (obrázek č. 4). Po instalaci nám pak stačí otevřít příkazový řádek a vepsat slovo „python“, podle kterého zjistíme, zda je *Python* nainstalován a také jakou verzi tohoto programovacího jazyka zrovna využíváme, což lze vidět na obrázku č. 5.

Obrázek č. 4: *Instalace Pythonu*



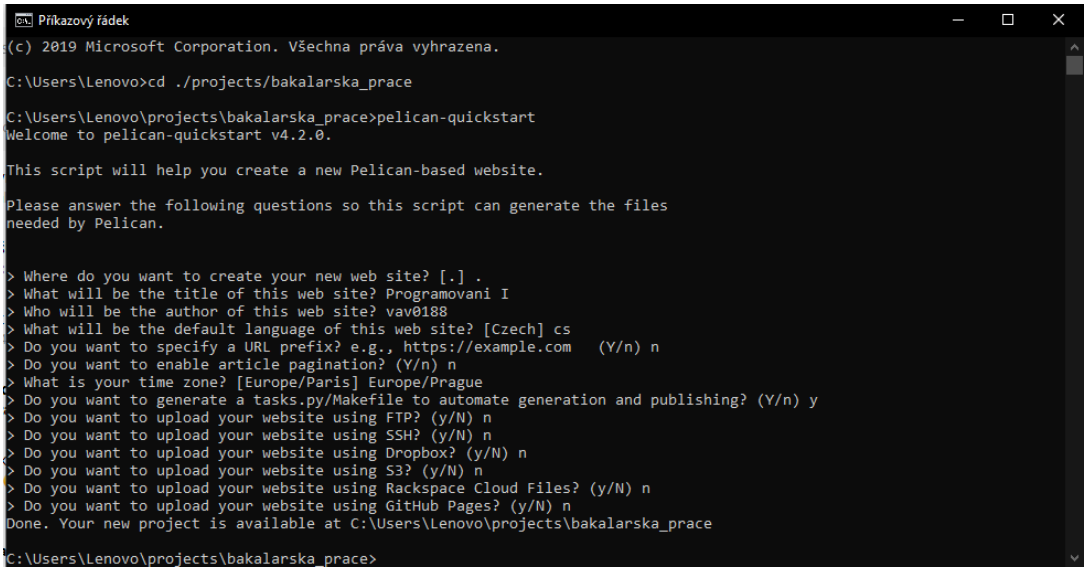
Obrázek č. 5: *Spuštění Pythonu*

Abychom se mohli postarat o rozšíření a využívání předem připravených modulů od komunity, byla vytvořena funkce *PIP*. Název je akronymem pro „Pip Installs Packages“. *PIP* je funkce, která se stará o správu modulů. Můžeme tak nainstalovat (pip install) či odinstalovat (pip uninstall) různá rozšíření. Od verze *Pythonu 2.7.9* a *3.4* je tato funkce defaultně instalována, v předešlých verzích je tak potřeba toto rozšíření stáhnout samostatně. [39]

5.1.2 Začátek s generátorem Pelican

Pelican není v základní verzi *Pythonu* instalován, musíme tak využít správce modulů *PIP* pro doinstalování tohoto generátoru. Znovu tak musíme využít příkazového řádku a v něm spustit tento příkaz „pip install pelican“. Následně je vhodné se rozhodnout, jakého značkovacího jazyku pro vytváření obsahu chceme využívat. V této práci byl zvolen *Markdown*. *Markdown* ale v porovnání s *reStructuredTextem* není při instalaci *Pelicanu* defaultně instalován, a tak musíme stáhnout i tento modul – „pip install Markdown“. V tomto momentě máme k dispozici funkční generátor, kterého budeme využívat. Nic dalšího k jeho funkci zatím není potřeba.

Protože je ale našim hlavním cílem vytvoření statických webových stránek, musíme započít se samotnou tvorbou webu. Jako první krok je nutné vytvořit kdekoliv na disku složku, do které budeme tento web generovat. Jelikož ale *Pelican* funguje pomocí příkazů, musíme se v *CMD* dostat právě do této složky, kde budeme spouštět všechny následné příkazy. První z nich je „pelican-quickstart“. Po spuštění tohoto řádku si musíme projít několika otázkami, které určí základní nastavení – viz obrázek č. 6.



Obrázek č. 6*: Začátek s Pelicanem*

Při vytváření nového projektu s *Pelicanem* se samotný generátor ptá na základní otázky. První položená otázka se dotazuje na místo generování statického webu. Jestliže nechceme měnit současnou cestu adresáře, napíšeme pouhou tečku. Následně napíšeme informace o názvu webu a jeho autorovi, určíme si jazyk, kdy „cs“ je zkratkou pro češtinu nebo „en“ pro angličtinu. Poté si určíme, zda chceme využít stránkování. Pokud se podíváme na stránky Šalouna, zjistíme, že zde není stránkování využito. Pokud by tomu tak bylo, vyskytla by se další otázka, a to kolik příspěvků chceme na stránce zobrazit. Poté se pokračuje určením časové zóny. V neposlední řadě nám *Pelican* umožní předem určit, jak budeme chtít web publikovat. V této práci není potřeba web nikam nahrávat.. Pokud bychom však chtěli,

tak se nás *Pelican* bude doptávat na další otázky. U *SSH* i *FTP* tak dojde k otázce, o jaký název serveru se jedná, cestu k místu, kde chceme tento web nahrát a jaké je naše uživatelské jméno. U *SSH* dostaneme otázku, jaký port chceme využít. Pro zbylé varianty se nás *Pelican* jen zeptá na cestu ke složce, u *Rackspace Cloud Files* dodáme i *API* klíč. Po těchto krocích máme vše předpřipravené. Obsah složky si můžeme prohlédnout na obrázku č. 7:

```
Directory of C:\Users\Lenovo\projects\bakalarska_prace

27.03.2020  19:06    <DIR>          .
27.03.2020  19:06    <DIR>          ..
27.03.2020  19:06    <DIR>          content
27.03.2020  19:06             2 659 Makefile
27.03.2020  19:06    <DIR>          output
27.03.2020  19:06               863 pelicanconf.py
27.03.2020  19:06               589 publishconf.py
27.03.2020  19:06             3 254 tasks.py
               4 File(s)          7 365 bytes
               4 Dir(s)  158 921 953 280 bytes free
```

Obrázek č. 7: *Složka po prvním spuštění Pelicanu*

Jak tedy vidíme, složka obsahuje dvě podsložky *content* a *output*. Do složky *content* ukládáme veškeré značkovací soubory, z kterých chceme vytvářet textový obsah, také v ní mohou být další podsložky, do kterých si vložíme veškeré multimediální soubory. Ve složce output se nám vždy vygeneruje náš požadovaný web včetně všech potřebných souborů a složek – *css*, *html*, *js* a veškeré doplňkové soubory. Poté zde vidíme soubor *Makefile*, který jsme si zvolili při samotném začátku (obrázek č. 6), jenž v sobě uchovává varianty o možné další publikaci webu. Na konci vidíme již pouze *Pythonovské* soubory. První z nich je *pelicanconf.py*. Tento soubor využíváme ve chvíli, kdy chceme měnit či přidávat jakékoliv nastavení k našemu webu, ale k tomuto souboru se vrátíme později. Druhý nastavovací soubor je *publishconf.py*, který své využití nachází ve chvíli, kdy chceme náš web někde nahrát, což v našem případě není potřebné. Poslední vygenerovaný soubor je *tasks.py*, díky kterému můžeme využívat vnitřního serveru k zobrazení vytvářeného webu.

5.1.3 Vytváření obsahu webu

V této práci je k vytváření obsahu webu užit odlehčený značkovací jazyk *Markdown*. Při pohledu na stránky k předmětu *Programování I.* musíme vytvořit celkem čtyři stránky, včetně vlastního indexu. Poslední pátou stránku – *Průběh semestru* – budeme vytvářet jako přidávání nových článků k této kategorii. Našim úkolem je tak vytvořit zatím tyto soubory – *prehled_pozadavku.md*, *uzitecne_odkazy.md*, *vyukove_texty.md* a v neposlední řadě soubor *home.md*, kterým si vytvoříme naší index stránku. Pokud bychom nevyužili této varianty, na indexu by se nám zobrazovaly pouze všechny nové přidané příspěvky, o což nestojíme.

První lehčí práce nás čeká právě na indexu. Našim úkolem je umožnit uživateli navštívit všechny stránky. Musíme ale začít standardní informací o stránce. Následně je potřeba vybrat titulek, datum publikace, jeho kategorii a taky název, pod kterým musíme náš nový index uložit. Celý nový index soubor je ukázán na výpisu kódu č. 6.

```
title: Start

category:start

date: 2018-08-20

save_as: index.html


###Programování I -- ZS 2018/2019


English version of page for Programming I is available and is actual
 for weekly labs,

see Programming 1 for English Students (will be updated).


[přehled požadavků/goals and requirements]({static}/category/prehled
_pozadavku.html)

…

<hr />

Během zimního semestru mám konzultační hodiny každý čtvrtek od 9.15
do 10.15 vmístnosti EA416.
```

Výpis kódu č. 6: *Část souboru home.md*

V této krátké ukázce si můžeme všimnout několika částí – část „save_as“ není vždy povinná, proto tuto variantu využíváme jen při našem indexu. Následně si všímáme třetí úrovně nadpisu „###“. Důležitým řádkem je také ukázka odkazu. Jelikož je našim cílem odkazovat na naši vygenerovanou stránku, musíme využít klíčového slova „{static}“ a napsání cesty k *HTML* souboru, která je ve složce *category*, poněvadž nastavujeme každému souboru jeho kategorii. V neposlední řadě si můžeme všimnout, že nemusíme vždy využívat značek *Markdown*, ale můžeme je promíchat i s přímými *HTML* značkami, jako je zde viditelná nepárová značka „<hr />“. Tímto způsobem tak nakonec vytvoříme všechny zbývající stránky, které nebudou mít žádné příspěvky.

Pro vytvoření stránky *prubeh_vyuky* budeme pracovat s blogovým pojetím webu. Nebudeme tak vytvářet jeden *md* soubor, do kterého vložíme všechen text, ale aby bylo možné příspěvky o průběhu výuky přidávat průběžně, budeme pro každý takový příspěvek vytvářet zvláštní soubor. Musíme si tedy dávat pozor, aby název kategorie byl neustále stejný. Pro ilustraci se podíváme na příspěvek *Aktualita 02* ve výpisu kódu č. 7. Můžeme tak vidět využití odrážek a také nahrání obrázku pro vlajku Velké Británie pro rozeznání anglického a českého textu, kdy v porovnání s odkazem využíváme vykřičník. Jak tedy vidíme, můžeme využívat jakýkoliv jazyk v jednom *Markdown* souboru.

```
Title: Aktualita 02
Date: 2018-12-05 10:20
Category: prubeh_semestru

###Aktuality - zápočtová písemka/Final Written Test
Informuji, že řádný povinný termín zápočtové písemky proběhne:

* pro kombinovanou formu studia v pátek 14. 12. 2018 v čase a místě
  naplánovaného tutoriálu,
* pro denní formu studia v pondělí 17. 12. 2018 v čase a místě vyhra
zené pro
  přednášku v Nové aule,

uvedené termíny jsou povinné, přihlásíte se na ně v systému Edison.
Bez přihlášení nedostanete zadání písemky.

![uk]({static}/imgs/uk-flag.png)English Group: Tuestay December 18,
2018 will be done the Final Written Test, the same place as the
lectures, time 2:15 p.m.
```

Výpis kódu č. 7: *Soubor 02_Aktualita.md*

Ve chvíli, kdy máme připraveny všechny potřebné soubory ve složce *content*, včetně *PDF* dokumentů, obrázků a dalších dodatkových souborů, můžeme v příkazovém řádku spustit příkaz „pelican content“. Po spuštění tohoto příkazu si můžeme všimnout úspěšnosti generování, případných chyb a jejich umístění a také počet vygenerovaných souborů. Následně tak stačí buď otevřít složku *output* a soubor *index.html* nebo v *CMD* využít příkazu „pelican --listen“ a otevřít si tento web na stránce http://localhost:8000. Po otevření této stránky si tak okamžitě všimneme, že web nemá žádný vzhled, což je zapříčiněno souborem *pelicanconf.py*.

5.1.4 Úprava nastavovacího souboru pelicanconf.py

Všechny dostupné úpravy budeme provádět v tomto souboru. V první řadě se jedná o povolení relativních odkazů. Tento dokument je na tuto variantu připraven, a tak stačí pouze smazat znak komentáře „#“ a ponechat řádek ve tvaru: „RELATIVE_URLS = True“.

Pokud se podíváme na naší vzorovou webovou stránku, tak zjistíme, že nemá žádné navigační menu. O to se budeme snažit i my, avšak s tím rozdílem, že budeme chtít uživateli umožnit přístup na stránku *index.html*, na což je *Pelican* opět připraven. Nezbývá nám tak nic jiného, než přidat prozatímní dva řádky – „DISPLAY_CATEGORIES_ON_MENU“ a „DISPLAY_PAGES_ON_MENU“, které nastavíme na „FALSE“. Nebudou se nám tak v navigačním

menu zobrazovat vůbec žádné odkazy. Poté nám zbývá jediné – přidat námi zvolené položky. V našem případě se bude jednat o „MENUITEMS = [('start', './'),]“. Toto nastavení je viditelné ve výpisu kódu č. 8.

Teď nám zbývá poslední část, a to zrušit veškeré odkazy na cizí stránky. To vše můžeme zařídit v blocích *Blogroll* a *Social widgets*. Stačí tak nechat závorky pouze prázdné, což lze také vidět ve výpisu kódu č. 8.

```
DISPLAY_CATEGORIES_ON_MENU = False

DISPLAY_PAGES_ON_MENU = False

MENUITEMS = [('start', './'),]

# Blogroll

LINKS = ()


# Social widget

SOCIAL = ()
```

Výpis kódu č. 8: *Ukázka části pelicanconf.py*

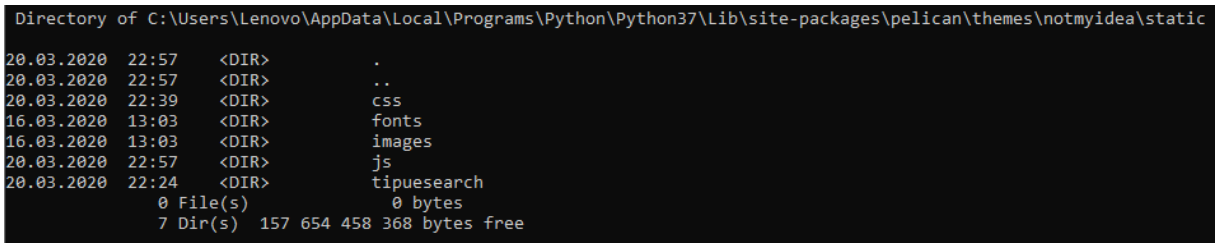
5.1.5 Přidání vyhledávacího pluginu

Na první pohled se může zdát, že naše stránky již mají vše. Avšak stále nám chybí jedna podstatná věc – vyhledávání. *Pelican* nám na stránkách https://github.com/getpelican/pelican-plugins zveřejnil veškeré vytvořené pluginy. Při prohlédnutí souboru *Readme.rst*, lze vidět všechny využitelná rozšíření. Pro náš případ zde existuje jediný vhodný – *Tipue Search*. *Tipue Search* je vyhledávácí plugin postavený na *JavaScriptu* a knihovně *jQuery*. Jedná se o velmi rychlý, open-source doplněk, který ke své práci vyžaduje pouze prohlížeč. [40]

Abychom mohli tohoto pluginu využívat, musíme si tyto doplňky stáhnout. Jedná se o samotný *Tipue Search* modul a také o plugin pro *Pelican* SSG. Modul si tak stáhneme z oficiálních stránek – https://www.tipue.com/search/tipuesearch.zip. U pluginu existuje více cest, můžeme si stáhnout úplnou kopii repozitáře nebo jen vybraný plugin pomocí prohlížeče. V našem případě využijeme druhé varianty a tento plugin stáhneme z tohoto odkazu – https://github.com/getpelican/pelican-plugins/blob/master/tipue_search/tipue_search.py. Následně je důležité také stáhnout verzi *jQuery* (v tomto případě 3.4.1 - https://code.jquery.com/jquery-3.4.1.js). K funkčnosti Python souboru potřebujeme doinstalovat ještě jedno rozšíření k využívání *JSONu*. Je to *BeautifulSoup*. Budeme ho stahovat z prostředí *CMD*, a to potvrzením příkazu „pip install beautifulsoup4“. Ve chvíli, kdy máme soubory stažené, můžeme začít s postupným upravováním a vkládáním.

V první řádě je potřeba vložit samotný modul k našemu *Pelicanu*. Ten je vždy uložen ve složce s Pythonem, pro ilustraci je zde uvedena cesta, která byla užita v této práci: „C:\Users\Lenovo\AppData\Local\Programs\Python\Python37\Lib\site-packages\pelican\“. Poté nám stačí nalézt složku *Theme*, která určuje vzhled našeho webu s přesnými šablonami. V našem případě využíváme defaultní vzhled *notmyidea*. Jelikož se jedná o statické *CSS* a *JavaScript* soubory (včetně

jQuery), které se při generování budou pouze kopírovat, vložíme celou složku *tipuesearch* do složky static i se staženou knihovnou *jQuery 3.4.1*, jak je možné vidět na obrázku č. 8.



Obrázek č. 8: *Výpis složek vybraného stylu*

Druhým krokem je vložit samotný plugin k našemu webu. V tomto případě ale nevkládáme Python pluginy do složky s *Pelicanem*, ale vkládáme je přesně naopak do složky, kde si generujeme web. Poté tak stačí vzít *tipue_search.py* soubor a vložit ho do složky *plugins*, kterou si předem vytvoříme.

Posléze je nutné upravit samotné šablony vzhledu. Je třeba definovat, odkud mají načíst nově přidané skripty, případně stylové soubory. Vrátíme se do složky *notmyidea*, ve které si vybereme složku *templates* a z ní následně základní soubor *base.html*. V tuto chvíli je potřeba přidat do hlavičky souboru vybrané statické soubory, které jdou vidět ve výpisu kódu č. 9.

```
<head>

        ...

        <link rel="stylesheet" href="{{ SITEURL }}/theme/tipuesearch
/tipuesearch.css">

        <script type="text/javascript" src="{{ SITEURL }}/theme/js/j
query-3.4.1.js"></script>

        <script type="text/javascript" src="{{ SITEURL }}/theme/tipu
esearch/tipuesearch_content.js"></script>

        <link rel="stylesheet" href="{{ SITEURL }}/theme/tipuesearch
/tipuesearch.css">

        <script type="text/javascript" src="{{ SITEURL }}/theme/tipu
esearch/tipuesearch_set.js"></script>

        <script type="text/javascript" src="{{ SITEURL }}/theme/tipu
esearch/tipuesearch.min.js"></script>

</head>


<body id="index" class="home">

...

        <form action="{{ SITEURL }}/search.html" onsubmit="return va
lidateForm(this.elements['q'].value);">
```

```
        <input id="tipue_search_input" type="text" name="q" placehol
der="Hledani">
        </form>
        </ul></nav>
        </header>
...
```

Výpis kódu č. 9: *Část souboru base.html*

V tomto výpisu si můžeme všimnout samotného template jazyka – *Jinja2*. Pomocí „{{ SITEURL }}“ zajistíme určitou dynamičnost obsahu. Jedná se o proměnnou, která odkazuje na potenciální doménu, jež můžeme nastavit v *pelicanconf.py*, ale v našem případě je tato proměnná prázdná. Druhou částí je přidání samotného textového pole v části dokumentu *body*, pomocí kterého budeme vyhledávat. V našem případě se nejvíce nabízí v porovnání s předlohou navigační menu. V kódu tak nalezneme oddíl „<nav></nav>“ a přidáme ho na konec této části v podobě formuláře – ukázka je viditelná ve výpisu kódu č. 9.

V tuto chvíli jsme si upravili první template soubor, avšak v návaznosti na tento soubor musíme vytvořit naprosto nový soubor – *search.html*. Jeho úkolem bude po potvrzení vyhledání řetězce zobrazit všechny nalezené výsledky. Budeme tak rozšiřovat základní soubor – *base.html*, který se musí v naší šabloně objevit. Jelikož tvoříme jen obsahovou část *HTML* dokumentu, musíme zde také uvést, odkud má náš plugin čerpat data. V našem případě se jedná o vygenerovaný *JS* soubor, který v sobě obsahuje objekt se všemi informacemi a daty našeho webu, které jsou uloženy ve formátu *JSON*. Následně tak bude v těchto objektech hledat shody s hledaným řetězcem. Abychom udrželi určitý styl našeho webu, využijeme předpřipravené části „<section>“ s identifikátorem *content* a třídou *body*. Poté pak pouze přidáme veškeré prvky, které jsou potřebné k vyhledávání a zobrazení výsledků – dva divy s předpřipravenými identifikátory a v neposlední řadě pár řádků skriptu, kterým budeme volat požadovanou vyhledávací funkci. Soubor search.html je pak zobrazen ve výpisu kódu č. 10.

```
{% extends "base.html" %}
{% block title %}{{ SITENAME }} Search {% endblock %}
{% block content %}
<script src="{{ SITEURL }}/tipuesearch_content.js"></script>

<section id="content" class="body">
<script>
    $(document).ready(function() {
        $('#tipue_search_input').tipuesearch({
            'show': 5,
            'newWindow' : true,
            'showTime':false
```

```
        });
    });
</script>
<div id="tipue_search_content">
    <div id="tipue_search_loading"></div>
</div>
</section>
{% endblock %}
```

Výpis kódu č. 10: *Template search.html*

Funkce *tipuesearch* má několik volitelných vlastností. Budeme tak využívat pouze show, díky které nastavíme, kolik příspěvků chceme na stránce zobrazit, dále budeme zobrazovat pouze celkový počet shodných příspěvků, avšak bez časového údaje (*showTime*). V neposlední řadě využijeme vlastnost *newWindow*, díky níž se odkazy vždy otevřou v nové liště prohlížeče.

V tuto chvíli máme všechny podpůrné materiály připraveny. Zbývá nám tak jen doplnit *pelicanconf.py* o nezbytné informace. V první řadě musíme *Pelicanu* říct, odkud a jaké pluginy má aplikovat. Do listu pro proměnnou *PLUGIN_PATH* napíšeme cestu k adresáři s pluginy. V tomto případě je to pouze složka *plugins* v hlavním adresáři stránek, stačí nám tak pouze tento název. Následně proměnné *PLUGINS* musíme říct, jaké pluginy má využívat. Kdybychom jich využívali více, byly by rozděleny čárkou, my však využíváme pouze *tipue_search*, a proto se v tomto listu objeví jen tento plugin. Jelikož jsme vytvářeli template *search.html*, musíme generátoru říct, že chceme využívat jak této šablony, tak i stránky *index.html*, proto je musíme do proměnné „DIRECT_TEMPLATES“ uložit. Všechny tyto úpravy lze vidět ve výpisu kódu č. 11.

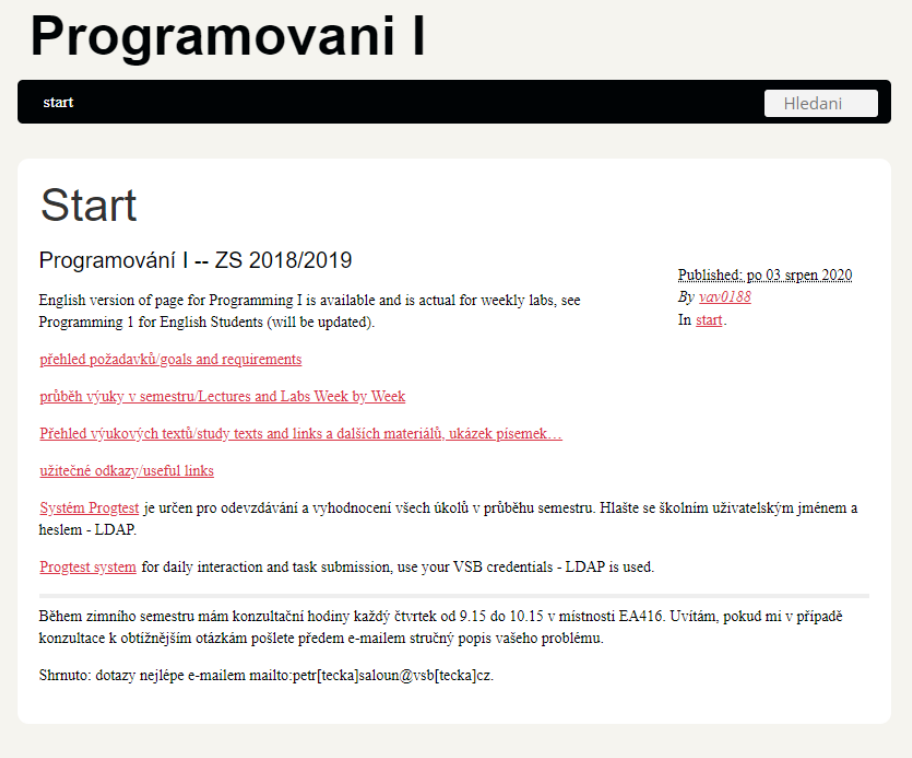
```
PLUGIN_PATHS = ['plugins']
PLUGINS = ['tipue_search']


DIRECT_TEMPLATES = (('index', 'search'))
```

Výpis kódu č. 11: *Pluginy v pelicanconf.py*

5.1.6 Ukázka vygenerovaného webu

V tento okamžik tak máme předpřipraveny všechny materiály a také upravené nastavení. Stačí nám do příkazového řádku vepsat „pelican content“, čímž vygenerujeme nové statické webové stránky. Na obrázku č. 9 se nachází ukázka statického indexu webu ke kurzu *Programování I.* Ukázání vyhledávací stránky a také jednoho příspěvku z průběhu výuky jsou viditelné v příloze D.



Obrázek č. 9: *Stránka index.html kurzu Programování I*

5.2 Stahování a využití statických webových stránek

V druhé případové studii jsme se rozhodli využít předem vytvořených statických stránek, které se zabývají konferencí *Data a znalosti*. Vzhledem k požadované bezpečnosti a záloze je vhodné, abychom všechny tyto materiály měli uloženy na nějakém statickém úložišti a nebyli tak závislí na samotné doméně a hostingu. Budeme tedy stahovat veškeré statické materiály, které potřebujeme k pouhé prezentaci obsahu.

Pro tuto práci budeme opět využívat programovacího jazyka *Python 3.7.6*., v jehož prospěch hraje zejména jeho jednoduchost, mnoho dostupných knihoven a také rychlé stahování. Budeme tak tvořit jednoduchý skript, jehož úkolem bude stahovaní veškerých potřebných souborů k funkčnosti stahovaného webu.

5.2.1 Web crawler

Web crawler budeme potřebovat z důvodu nutnosti prohledávání celých *HTML* dokumentů. Abychom mohli stahovat celé weby, musíme si uvědomit, jakých značek si v souboru *HTML* budeme všímat. Pro nás tak bude nejdůležitější z webů vytáhnout tyto značky:

- „<a>“ – pro veškeré podstránky webu, včetně přidaných souborů, jako je *pdf*, *zip* či *pptx*
- „<link>“ – zejména pro stylové soubory
- „<img>“ – pro všechny obrázky
- „<script>“ – pro naše *JavaScript* soubory, o které nechceme přijít
- „<frame>“ – starší weby jsou skládány pomocí odlišných rámů, které jsou zobrazovány na jediné *HTML* stránce

K funkčnosti našeho *crawleru* tak využijeme importování několika knihoven a modulů. Ve velké míře tak budeme pracovat s balíčkem *urllib*, jehož úkolem je otevírání *URL* odkazů s následnými akcemi. [41] V závislosti na *urllib* budeme využívat jeho tři moduly:

- *request* – pro samotné přistupování a otevírání stránek (využití funkce *urlopen*);
- *parse* – pro práci se samotným odkazem (*urljoin*);
- *error* – v situaci, kdy nemáme přístup k požadované stránce.

A jelikož potřebujeme zejména náš dokument „rozebrat“, využijeme *BeautifulSoup*. *BeautifulSoup* je knihovna, která je vhodná k parsování *HTML* nebo *XML* dokumentů. Má tak několik předpřipravených funkcí k analýze takového dokumentu a my, jako programátoři, si ušetříme mnoho hodin času. [42] V tomto případě je pro nás nejvhodnější „html.parser“, který se postará o veškerou práci v pozadí se značkami, jež budeme chtít vyhledávat.

To nejdůležitější tak máme naimportováno a můžeme začít se samotnou *crawl* funkcí (viditelná ve výpisu kódu č. 12). Tato funkce má několik vstupních parametrů: *pages* (jedná se o list, který má v sobě uloženy veškeré požadované odkazy), *type* (tímto parametrem nastavíme, jakou značku chceme vyhledávat v dokumentu) a v neposlední řadě *name* (pomocí něhož nastavíme, jaký atribut chceme ze značky vybrat pro požadovaný odkaz). V první řadě tak musíme pomocí cyklu *for* projít všechny stránky, které jsou uloženy v listu *urls*. Poté je potřeba pro každou takovou stránku, která je jedinečná a není již ve výsledném poli obsažena, vykonat několik operací. Stránku si tak pomocí funkce *urlopen* otevřeme, pokud nám nebude vyvolána výjimka, bude kód směle pokračovat. V tuto chvíli přichází na řadu *BeautifulSoup*. Do této funkce předáme dva parametry – první bude navracený objekt z funkce *urlopen*, který může vypadat například takto: „<http.client.HTTPResponse object at 0x000001A474F33F88>“, druhým parametrem funkci řekneme, že chceme využívat parser pro *HTML* dokumenty. Tento argument sice nastavovat nemusíme a funkce by minimálně na našem stroji mohla fungovat, ale nemáme zajištěnou funkcionalitu i na jiných zařízeních, což by mohl být problém. Poté, jen z celého souboru uloženém v proměnné *soup*, vybereme tu značku, kterou jsme si sami určili a získáme tak list všech těchto značek. Protože je našim primárním cílem získat odkazy na požadované soubory, budeme tento list procházet značku po značce za využití *dictionary* (slovníku), který funguje na principu „klíč:hodnota“. Takto tedy získáme všechny atributy, které značka obsahuje. Tímto způsobem ale získáme pouhou interní cestu na webové stránce. My potřebujeme pracovat s celým odkazem, proto musíme propojit zadanou *url* z listu stránek a spojit ji s touto interní cestou. K tomu je připravená již dříve zmíněná funkce *urljoin*. I když máme „nějaký“ odkaz k dispozici, je potřeba zjistit, zda ho potřebujeme celý. V dnešní době se hojně využívají tzv. single-page website, kdy je veškerý obsah psán v jednom *HTML* souboru. Proto jsme se rozhodli zbavit určité jedinečnosti samotných odkazů na stránce, jelikož odkaz „https://hi.kkui.fei.tuke.sk/daz2019/index.html#section-ajenda“ je úplně stejný, jako odkaz „https://hi.kkui.fei.tuke.sk/daz2019/index.html“ a my tak ulehčíme samotnému stahovacímu procesu.

```
def crawl(pages, type, name):
    urls = []
    for page in pages:
        if page not in urls:
            try:
                c = urllib2.urlopen(page)
            except:
                continue
            soup = BeautifulSoup(c,"html.parser")
            links = soup(type)
            for link in links:
                if name in dict(link.attrs):
                    url = urljoin(page, link[name])
                    if url.find("'") != -1:
                        continue
                    url = url.split('#')[0]
                    if url not in urls:
                        urls.append(url)
        pages = urls
    return urls
```

Výpis kódu č. 12: *Funkce crawl*

V tuto chvíli máme dokončený *crawler*. Není nijak omezený na značky, kterými se budeme chtít zabývat, proto tuto funkci budeme využívat na všechny výše zmíněné.

5.2.2 Jména souborů

Našim úkolem je zachovat správnou hierarchii webu. Je proto potřeba, aby volená jména stahovaných souborů byla správná. Taktéž nesmíme zapomenout na to, že i když soubory stáhneme, musíme je mít ve správných složkách a podsložkách. K tomu využijeme knihovnu *os*.

Knihovna *os* je pro nás vhodná z toho důvodu, že nám umožňuje využívat „operating system dependent functionality.“ [43] Je také důležité zmínit, že funkce operačního systému, které nám jsou umožněny, závisí na používaném operačním systému. Můžeme tak zjistit, na jaké pozici v paměti se nacházíme, také informace o procesech nebo zde lze například jakkoliv manipulovat se soubory.

Opět si tak vytvoříme funkci, která bude mít několik vstupních proměnných. V tomto případě se bude jednat o samotnou *URL*, se kterou momentálně pracujeme, a také je potřeba vložit rok uskutečnění dané konference. V prvním kroku jen zjistíme, zda je posledním znakem lomítko, pokud není, jednoduše si ho přidáme. Následně si musíme nalézt všechna lomítka. Samotná webová adresa

totiž začíná s formátem „http(s)://“, a tak se musíme zbavit této části. Poté také víme, že samotný web může či nemusí být uložen v nějaké složce. Této části se tak můžeme pohodlně zbavit snadnou kontrolou, jestli od posledního výskytu lomítka, existuje nějaké další. Poté se pokračuje s budováním samotného jména souboru a také vytvoření všech podsložek. V případě, že předáváme této funkci takovou adresu, která v sobě nemá jediný soubor (všechny soubory v sobě obsahují přímo před koncovku tečku), víme, že se jedná o jakoukoliv složku z webu. A jelikož se jedná o odkaz, tak také víme, že v tomto případě prohlížeč vždy zobrazí stránku *index.html*, kterou tak přidáme k cestě k souboru. Poté nám jen stačí pomocí funkce „makedirs“ z knihovny *os* vytvořit všechny složky. Pokud však soubor tečku po posledním lomítku obsahuje, pokračuje funkce dál. Pokusí se opět nalézt pozici úplně posledního lomítka. V případě, kdy má adresa v sobě jakýkoliv soubor, musíme si zajistit, že cesta bude právě bez názvu takového souboru, abychom mohli vytvořit pouze všechny složky. Jestliže bychom to nezajistili, může se stát, že se nám na disku vytvoří složky, jako je například: „style/styl.css/“, o což nestojíme, my chceme pouze „style/“ a poté stáhnout a uložit soubor *styl.css*. V dalším kroku se opět zkontroluje, zda taková cesta již neexistuje, pokud tedy neexistuje, vytvoří se a funkce vrátí zpět celou adresu k požadovanému souboru, který se bude následně stahovat. Samotnou funkci určenou pro vytvoření cesty a jména souboru je vypsána ve výpisu kódu č. 13.

```
def getname(url,rok):
    if url[-1] == "/":
        http = url
    else:
        http = url+"/"
    first = http.find("/")
    second = http.find("/", first + 1)
    last = http.find("/", second + 1)
    txt = "historie/"+rok+"/"
    if (http[last:]).find(".") > 0:
        http = http[:-1]
    if http.find("./") > 0:
        txt = "../"
    if (http[last:]).find(".") < 0:
        tmp = 0
        if http[last] == "/":
            tmp = 1
        if not os.path.exists(txt+http[last+tmp:]):
            try:
                os.makedirs(txt+http[last+tmp:])
            except:
```

```
                return txt+http[last+tmp:]+"index.html"
        return txt+http[last+tmp:]+"index.html"
    else:
        soubor = http
        if http.find("/",last+1) > 0:
            next = http.rfind("/",last+1)
            if http[next:].find("."):
                soubor = http[:next+1]
            if not os.path.exists(txt+http[last+1:next]):
                try:
                    os.makedirs(txt + soubor[last+1:])
                except:
                    return txt+http[last+1:]
            return txt+http[last+1:]
        return txt+http[last+1:]
```

Výpis kódu č. 13: *Funkce getname*

5.2.3 **Stažení souboru**

V první řadě je nutné si uvědomit, čemu se musíme při stahování vyvarovat a jaké funkce budeme potřebovat. Pokud se podíváme na předaný název, proměnná stále může obsahovat značky spojené s *query stringem*. *Query string* je část *URL* odkazu. Využívá se ve chvíli, kdy se posílají určitá data na server a většinou v tomto případě na základě požadavku i zpět data odesílá. Vypadá tak například takto: „https://ukazka.cz/zadost.php?jmeno=pavel&prijmeni=vavruska“. Jak vidíme, tento odkaz má několik specifických znaků, ale vždy, když je *query string* použit, obsahuje v sobě otazník. And (&) má funkci pro spojování takových předávaných parametrů. [44]

V situaci, když víme, odkud máme vzít veškeré odkazy a připravit jim cestu, můžeme začít tyto soubory postupně stahovat. Stačí nám tak pouhý odkaz (*file*) a jejich skutečný název (*name*) vzhledem k hierarchii stahovaného webu. Do této funkce tak budeme vkládat tyto dvě informace. Dále se pokusíme stáhnout požadovaný soubor. Prvně je tak opět potřeba stránku otevřít prostřednictvím funkce *urlopen*. Dalším krokem je skutečně prozkoumat samotný název ukládaného souboru. Nechceme totiž, aby se v něm objevili jakékoliv značky spojené s *query stringem*, protože ve chvíli, kdybychom je udrželi v názvu našeho souboru, pracovalo by se i s takovým názvem souboru, což v našem případě chceme pouze ve chvíli, kdy víme, že tento odkaz v sobě uchovává „id“ (pro generování specifické stránky v *PHP* dokumentu) nebo „media“ (určuje cestu pro mediální soubory). V situaci, kdy v odkazu otazník je, pokusíme se nalézt tato dvě slova. Pokud nalezneme pozici pro „id=“, vygenerujeme si nové jméno takové stránky, která se nachází za rovnítkem. Stačí nám tak zjistit pozici posledního lomítka v cestě k souboru, za kterou přidáme právě reálný název stránky z části za „id=“. Pro „media=“ zvolíme stejný přístup, jen s tím rozdílem, že nemusíme názvu přidávat koncovku. Pokud název souboru

obsahuje otazník bez těchto dvou klíčových slov, bude se název souboru brát pouze do možného výskytu otazníku pomocí funkce *find*. Pokud se nalezne otazník, pracujeme s názvem takového souboru po tento znak, pokud ne, nemusíme činit žádné úpravy. Pokud se nenalezne otazník, pokračujeme v práci dál. Zejména nesmíme zapomenout na takovou situaci, kdy je web skládán z dříve zmíněných *PHP* souborů bez otazníku. Samotná struktura dokumentu je sice stejná, jako *HTML*, avšak soubor *PHP* spustíme pouze za využití serveru. Z tohoto důvodu tak v situaci, kdy je v názvu souboru *.php*, nahradíme tuto koncovku za *.html*. Na dalším řádku tak vidíme otestování takové cesty souboru (do funkce je předávána pomocí parametru *name*). Pokud takový soubor na disku neexistuje, můžeme tento soubor stáhnout a vytvořit. Pro vytvoření souboru využijeme funkci *open* ve spojení s klíčovým slovem *with*. *With* je vhodné ve chvíli, kdy pracujeme se soubory z důvodu automatického zavření souboru po dokončení bloku. U funkce open musíme nastavit pár vlastností. V prvním případě se jedná o cestu k samotnému souboru, druhým parametrem pak označujeme, v jakém režimu chceme s touto funkcí pracovat. V našem případě ji chceme binárně zapisovat, proto nastavíme „wb“ (write a binary). Tento režim nám také umožňuje v případě nenalezení otevíraného souboru samotný soubor nově vytvořit. Poté jen na tento soubor zavoláme funkci *write* a zapíšeme veškerý obsah, který je nám předán prostřednictvím funkce *read*, volánou na objekt, který nám *urlopen* vrací. V případě chyby využijeme bloku *except* a funkci pouze ukončíme. Tato funkce je vyobrazena ve výpisu kódu č. 14.

```
def downloadfile(file, name):
    try:
        file = urllib2.urlopen(file)
        qst = name.find("?")
        nm = name
        if qst > 0:
            posId = nm.find("id=")
            posMed = nm.find("media=")
            if posId > -1 or posMed > -1:
                if nm.find("&") > -1:
                    posAnd = nm.find("&", posId + 3)
                    if posId > -1:
                        naa = name[:name.rfind("/") + 1] + nm[posId
+ 3:posAnd] + ".html"
                    elif posMed > -1:
                        naa = name[:name.rfind("/") + 1] + nm[posMed
 + 6:posAnd]
                else:
                    if posId > -1:
                        naa = name[:name.rfind("/") + 1] + nm[posId
+ 3:] + ".html"
```

```
                    elif posMed > -1:
                        naa = name[:name.rfind("/") + 1] + nm[posMed
+ 6:]
                if not os.path.exists(naa):
                    with open(naa, 'wb') as output:
                        output.write(file.read())
            else:
                naa = nm.replace(".php", ".html")
                if not os.path.exists(naa):
                    with open(naa[:qst], 'wb') as output:
                        output.write(file.read())
        else:
            naa = nm.replace(".php",".html")
            if not os.path.exists(naa):
                print("soubor: "+naa)
                with open(naa, 'wb') as output:
                    output.write(file.read())
    except:
        pass
```

Výpis kódu č. 14: *Funkce downloadfile*

5.2.4 Finální část stahování

Než se dostaneme k samotnému stažení všech potřebných dokumentů, musíme uživateli umožnit přijatelný vstup. Postupně se ho tak budeme ptát na požadovanou stahovanou stránku (musí se jednat o první stránku – *index*) a také na rok konference. Následně bude provedena kontrola zadaných údajů. *URL* musí být v požadovaném formátu kontrolovaným pomocí funkce *checkUrl*, do které vkládáme *url* a poté provedeme kontrolu, zda jsou všechny části odkazu zapsány ve správném formátu a nic nechybí (proto využití funkce *all*, která vrátí *True* ve chvíli, kdy jsou všechny podmínky splněny). Druhou podmínkou, pro správnost odkazu, je právě jeho existence, z toho důvodu se stránku pokoušíme otevřít. Kontrola roku konference musí být v rozmezí let 1900 a současného roku. Rok získáme za využití naimportování knihovny *datetime* a funkcí *date.today().year*. Tato kontrola a vyhodnocení je viditelná ve výpisu kódu č. 15.

```
def checkUrl(url):
    try:
        result = urlparse(url)
        return all([result.scheme, result.netloc, result.path])
    except:
        return False
check = False
check2 = False
ind = ""
year = ""
print("Vlož prosím webovou adresu s jeho hlavní stránkou (https://ww
w.vsb.cz/)")
while(check != True):
    url = input()
    check = checkUrl(url)
    ind = url
    if check == False:
        print("Zadej správný tvar webové adresy.")
    if check == True:
        try:
            c = urllib.urlopen(ind)
            check = True
        except:
            print("Taková webová stránka neexistuje.")
            check = False
while(check2 != True):
    print("Vlož rok konference")
    rok = input()
    yr = datetime.date.today().year
    try:
        val = int(rok)
        if val > 1900 and val <= yr:
```

```
            check2 = True
            year = str(rok)
    except:
        check2 = False
```

Výpis kódu č. 15: *Kontrola vstupu uživatele*

V současné situaci jsme si ukázali, jaké funkce budeme při tomto stahování webů potřebovat. Teď nám pouze stačí tyto funkce vhodně využít. Uživatel nám zadal *index* konference (proměnná *ind*) a její rok (*year*). Prvně si upravíme předaný odkaz. Pokud odkaz nekončí lomítkem (poslední znak není lomítko a v odkazu se nenachází předaný soubor), přidáme mu ho sami. Také chceme zajistit, abychom používali a stahovali pouze ty soubory, které mají stejnou doménu (nebudeme tak stahovat soubory externí a ty soubory, které nejsou uloženy na požadovaném hostingu). K zjištění této domény využijeme funkci *urlparse* z knihovny *urllib.parse*, které jen předáme odkaz ke stahovanému webu. Následně tak získáme čistý odkaz bez jakýchkoliv podsložek, které ale chceme získat z důvodu stahování přesně určených souborů samotného webu. Prvně si ale v tomto odkazu nalezneme pozici pro „www.“, případně pro „://“, abychom věděli, kde nám začíná doména, a následně nalezneme i poslední tečku v odkazu, abychom věděli, kde doména končí. Tuto část domény předáme funkci *updateLinks* (výpis kódu č. 16), která nám následně vrátí celkovou doménu i se správnými podsložkami na cestě k *index.html*. Po získání domény si zavoláme funkci *crawl*, díky které zjistíme první potřebné odkazy a pododkazy zpracovávaného webu (web se může skládat buď ze z tagů „a“ nebo může být složen pomocí rámů – „frame“). Tyto listy sloučíme, přidáme k nim i předaný indexový soubor uživatele a pomocí funkce set z něj vytvoříme jedinečný set bez duplikátů. Tento set odkazů postupně projdeme, jednoduchou podmínkou zjistíme, zda se jedná o odkaz na webovou stránku (na konci je buď *.html/.php* nebo pouhé lomítko), poté zkontrolujeme, zda obsahuje zadanou doménu, a v případě, že jsou podmínky vyhodnoceny kladně, zavoláme na takovou stránku funkci *downloadAll*.Vše je možné vidět ve výpisu kódu č. 16.

```
if ind.rfind("/")+1 != len(ind) and (ind[ind.rfind("/"):]).find(".")
 < 0:
    ind = ind+"/"
parsed = urlparse(ind)
url = '{url.scheme}://{url.netloc}/'.format(url=parsed)
domain = ""
if url.find("www.") > -1:
    first = url.find("www.")
    last = url.rfind(".")
    domain = ind[first + 4:last]
else:
    first = url.find("//")
    last = url.rfind(".")
    domain = ind[first+2:last]
dom = updateLinks(ind, year, domain)
```

```
urls = [ind]
urls2 = crawl(urls, "a", "href")
urls2.insert(0,ind)
urls3 = crawl(urls,"frame","src")
urls = urls2.extend(urls3)
urls2 = set(urls2)
for page in urls2:
    if page.find(".html") > -1 or page[-
1] == "/" or page.find(".php") > -1:
        if page.find(dom) > -1:
            downloadAll(page,dom,year)
```

Výpis kódu č. 16: *Ukázka získání domény a průchodu odkazů*

Funkci *downloadAll* předáme celkem tři parametry – *ind*, *domain* a *year*. *Ind* v sobě uchová samotný odkaz na stránku, *domain* si uloží doménu stránky a *year* bude opět proměnná pro rok konference. Následně zavoláme funkci *crawl* pro získání všech dalších pododkazů (buď jako odkaz ve značce „<a>“ nebo jako rámec „<frame>“), přidáme předaný odkaz a vytvoříme sloučení těchto listů bez duplikátů. Následně za pomoci cyklu projdeme tyto odkazy. Nesmí se v nich vyskytovat „mailto“ (využití při emailových odkazech) a musí obsahovat doménu webové stránky. Jelikož se v první řadě jedná pouze o odkazy, provedeme jednoduchou kontrolu – pokud odkaz končí lomítkem, přidáme mu také *index.html*. V případě, kdy máme k dispozici odkaz bez lomítka na konci a v části od posledního lomítka po konec řetězce se nenachází ani tečka (označení, že se jedná o soubor), přidáme tak na konec tohoto neukončeného odkazu lomítko s *index.html*. Po tomto kroku můžeme předaný odkaz stáhnout pomocí funkce *downloadfile*. Po stažení ale funkce nekončí. Jestli ve svém názvu uchovává *.html* nebo *.php*, můžeme v tomto odkazu nacházet další důležité soubory. Za využití funkcí *crawl* si postupně do listů uložíme odkazy, kaskádové styly, obrázky a *JS* skripty. Tyto listy pak postupně projdeme. U odkazů pouze zjistíme, že nekončí koncovkou *html* a *php*, že mají v sobě zadanou doménu a také, že se nejedná o emailový odkaz. Pokud se však o odkaz na další stránku jedná, provedeme kontrolu, zda taková stránka již není v listu *urls1* a také, zda takový soubor již není na disku. Pokud je podmínka vyhodnocena kladně, předáme tuto adresu znovu funkci *downloadAll*. U dalších průchodů setů pro další soubory provedeme pouze kontrolu domény. Tato funkce je k vidění ve výpisu kódu č. 17.

```
def downloadAll(ind,domain,year):
   pagelist = [ind]
   urls = crawl(pagelist, "a", "href")
   urls5 = crawl(pagelist, 'frame', 'src')
   urls.extend(urls5)
   urls.insert(0,ind)
   urls = list(set(urls))
   for page in urls:
      if page.find("mailto") == -1:
          if page.find(domain) > -1:
```

```
                name = getname(page,year)
                if not os.path.exists(name):
                    downloadfile(page, name)
                    if name[-1]=="/":
                        name += "index.html"
                    if name.find(".",name.rfind("/")) == -1 and
name.rfind("/")+1 != len(name):
                        name += "/index.html"
                    if name[-5:] == ".html" or name[-4:] == ".php":
                        pg = [page]
                        urls1 = crawl(pg, "a", "href")
                        urls2 = crawl(pg, "link", "href")
                        urls3 = crawl(pg, 'img', 'src')
                        urls4 = crawl(pg, 'script', 'src')
                        urls1 = set(urls1)
                        urls2 = set(urls2)
                        urls3 = set(urls3)
                        urls4 = set(urls4)
                        for page in urls1:
                              if page.find(domain) > -1 and
page.find(".html") == -1 and page.find(".php") ==-1 and
page.find("mailto") == -1:
                                 downloadfile(page, getname(page, year))
                             elif page.find(domain) > -1 and
(page.find(".html") > -1 or page.find(".php") > -1) and
page.find("mailto") == -1:
                                 nm = getname(page, year)
                                 if nm[-1] == "/":
                                     nm += "index.html"
                                 if nm.find(".", nm.rfind("/")) == -1
and nm.rfind("/") + 1 != len(nm):
                                     nm += "/index.html"
                                 if page not in urls and not
os.path.exists(nm):
                                       downloadAll(page, domain, year)
                        for page in urls2:
                            if page.find(domain) > -1:
                               downloadfile(page, getname(page, year))
                        for page in urls3:
                            if page.find(domain) > -1 and
page.find("link") < 0:
                               downloadfile(page, getname(page, year))
                          for page in urls4:
                             if page.find(domain) > -1:
                               downloadfile(page, getname(page, year))
```

Výpis kódu č. 17: *Funkce downloadAll*

5.2.5 Ukládání odkazů konferencí

Pro pozdější propojování konferencí pomocí statických odkazů, potřebujeme všechny tyto adresy ukládat v externím souboru. V našem případě využijeme *pythonovský* soubor, ve kterém budeme mít dva listy – *links* a *links2*, abychom s nimi mohli dále pracovat. List *links* si uloží část adres odkazů a *links2* si uchová statické cesty k této webové stránce.

K této funkci využijeme funkce *updateLinks*, do které budeme předávat samotný index stránky, rok konference a doménu. Prvně si ale musíme provést kontrolu dříve zmíněných listů. Víme, že statické weby ukládáme do složky historie podle roku uskutečnění konference. Musíme tak projít všechny tyto záznamy v listu *links2* a v případě, že se v něm nachází zadaný rok, smažeme všechny tyto adresy z obou ukládaných listů. Abychom mohli s těmito listy pracovat a mohli je poté opět uložit, využijeme lokálních proměnných *lnk* a *lnk2*. V proměnné *tmp* se bude nacházet číslo, které nám určí ukončení domény (od této pozice se totiž bude nacházet doména prvního řádu a všechny další podsložky, pro přesné určení adresy konference). Proměnná *address* v sobě tak má jak doménu, tak i zbytek odkazu. V proměnné *ad* se bude nacházet část té adresy, která začíná prvním lomítkem (již bez jakékoliv zmínky domény a domény prvního řádu) až po poslední výskyt lomítka nebo konec této proměnné. V případě, že na disku máme složku konference ze zadaného roku (pomocí funkcí z knihovny *os* zjistíme existenci takové cesty), smažeme ji za pomocí funkce *shutil.rmtree* (nutné importování knihovny *shutil* a *os*). Do lokální proměnné *lnk* si uložíme proměnnou *address* i s posledním lomítkem a do *lnk2* si vložíme cestu k souboru na disku. Poté nám stačí do souboru *all_links.py* uložit oba tyto aktualizované listy a vygenerovat souhrnný index konference, pomocí kterého máme přístup ke všem ročníkům. Funkce nám pak vrátí úplnou doménu webové stránky včetně podsložek. Celý výpis této složky je ve výpisu kódu č. 18.

```
def updateLinks(ind,year,domain):
    pos = -1
    for i in links2:
        pos += 1
        if i.find(year) > 0:
            del links[pos]
            del links2[pos]
    txt = "../" + year + "/"
    lnk = links
    lnk2 = links2
    tmp = ind.find(domain) + len(domain)
    address = domain + ind[tmp:]
    ad = address[address.find("/") + 1:address.rfind("/") + 1]
    if os.path.exists("historie/"+year):
        try:
            shutil.rmtree("./historie/"+year)
        except:
            print("Složka nemusela být správně smazána.")
    lnk.append(address[:address.rfind("/")]+"/")
    lnk2.append(txt+ad+"index.html")
```

```
    with open("all_links.py", "w") as file:
        s = "links = " + str(lnk) + "\n" + "links2 = " + str(lnk2)
        file.write(s)
    generateIndex()
    return address[:address.rfind("/")]+"/"
```

Výpis kódu č. 18: *Ukládání odkazů do listů links a links2*

5.2.6 Vytvoření vstupního indexu stahovaných konferencí

Tuto funkci *generateIndex* vytváříme z toho důvodu, abychom uživateli umožnili přístup ke všem uloženým ročníkům konference. Nám tak stačí postupně projít všechny prvky v listu *links2*, kde máme uloženy tyto statické cesty, a vygenerovat si značky „div“ pro následné stylizování a také „a“, díky čemuž se dostaneme na zadaný web. Jelikož v *links2* máme jak rok, tak i dvě tečky (jsou vhodné při nahrazování odkazů v souboru *soubory.py*), projdeme si pomocí cyklu všechny tyto záznamy, nalezneme si dvě první lomítka, mezi kterými máme uložený rok konference, jenž použijeme pro popisek tagu *a*. Po vygenerování všech položek "div" si uložíme do souboru index.html všechen obsah *HTML* dokumentu. Tento kód je viditelný ve výpisu kódu č. 19.

```
def generateIndex():
    divs = ""
    for i in links2:
        frs = i.find("/")
        scd = i[frs+1:].find("/")
        year = i[frs+1:frs+scd+1]
        divs +='<div class="link"><span class="button"><a
target="_blank" href="historie'+i[i.find("..")+2:]+'"> Ročník
'+year+'</a></span></div>\n'
    text = """<!DOCTYPE html>
    <html>
        ..."""+divs+"""...
    </html>"""
    with open("index.html", "w+", encoding="UTF-8") as file:
        file.seek(0)
        file.truncate()
        file.write(text)
```

Výpis kódu č. 19: *Funkce pro generování stránky index.html*

5.2.7 Modifikace HTML dokumentů pro statické využití

V tuto chvíli sice máme uložené veškeré historické ročníky požadované konference, ale stále nám chybí tyto složky projít a najít v nich všechny *HTML* dokumenty. Musíme se totiž postarat o to, aby samotné odkazy byly spustitelné a také, aby byly samotné ročníky navzájem proklikatelné.

V první řadě tak potřebujeme získat všechny takové dokumenty. To lze provést za opětovného použití knihovny *os* a její funkce *walk*. Tečka umístěna uvnitř nám umožní průchod této stromové struktury do všech složek za využití cyklu se třemi proměnnými – *i* (dostaneme cestu s kořenem), *j* (dostáváme složky) a *k* (název souborů). My chceme využít cestu, a tak proměnné *j* a *k*, nevyužijeme. Tato proměnná nám tedy vrací cestu ve formátu „./cesta“. Jelikož známe současné umístění všech historických ročníků, stačí nám k vytvoření úplné cesty nalézt tečku, kterou následně ke zpracovávání cesty ke složkám nepotřebujeme, a od této pozice využít tuto cestu. Aby cesta byla správná, provedeme i případnou náhradu lomítek. V tuto chvíli musíme nastavit novou pracovní pozici podle složky, kterou chceme prohledávat. Využijeme tak knihovny *glob* a její stejnojmenné funkce k nalezení všech souborů, které končí příponou *.html*. Poté nám stačí pomocí cyklu projít všechny soubory a ty si přidat do listu *files*. Do proměnné si uložíme celkovou délku zadané cesty k historickým ročníkům.

V tuto chvíli jsme si nalezli všechny soubory, se kterými chceme pracovat. Musíme si je tak postupně projít. V proměnné *f* bude uložena cesta k souboru pouze od jeho roku uspořádání. Víme tedy, že každý ročník je uložen ve složce s jeho rokem, proto nám stačí nalézt první a druhé lomítko, mezi kterými se tento rok nachází. Abychom pracovali se správnou pozicí v dříve zmíněných listech *links* a *links2*, je potřeba nalézt pozici zpracovávaného roku konference, díky kterému zjistíme, kolik podsložek je potřeba projít k souboru *index.html* zpracovávaného roku konference pomocí přesného počtu lomítek k tomuto index souboru (proměnná *count*). Do proměnné *dr* tak budeme přidávat tolik skoků, které zapisujeme v podobě „../“, kolik je potřeba vzhledem k hodnotě proměnné *count*. Následně nám stačí vzít upravenou část cesty zpracovávaného souboru, nalézt první a druhé lomítko, se kterými nijak nepotřebujeme pracovat, a posléze si určit celkový počet zbývajících lomítek (*count2*), které určí přesnou pozici souboru v hierarchii. Rozdílem hodnot *count2* a *count* si určíme, kolik skoků vůči indexu musíme provést (budeme přidávat do proměnné *dirs* „../“ v závislosti na rozdílu *count3*).

Prvně se postaráme o náhradu odkazů v souborech vůči propojení ročníků mezi sebou. Budeme tak procházet list *links*, kde máme tyto odkazy uloženy. Prvně si tak binárně přečteme zpracovávaný soubor za využití funkce *open*. Při binárním čtení nemusíme znát kódování souboru, pro další práci je to však potřebné. Využijeme tak knihovny *chardet* a její funkce *detect*, která nám prozradí, o jaké kódování se jedná. Při dalším otevření tohoto souboru tak můžeme pracovat čistě s jeho obsahem a správným kódováním. Při čtení *HTML* si do proměnné uložíme jeho veškerý obsah, abychom následně tuto proměnnou mohli předat funkci *BeautifulSoup* s *HTML* parserem. Víme tedy, že se vždy jedná o odkazy, které se značí tagem „a“. Nalezneme je všechny a následně je budeme postupně procházet. Pokud má odkaz nějakou adresu, je uložena v atributu „href“. Jestliže tento atribut má, funkce *find* nám umožní zjistit, zda se odkaz z listu *links* nachází v této části. Pokud tam je, nastavíme tagu „a“ úplně nový atribut *href* se správnou cestou k uloženému statickému souboru za využití skoků *dr*, *dir* a také hodnotu v listu *links2* ke správnému indexovému souboru ročníku nahrazovaného odkazu. V neposlední řadě nesmíme zapomenout na obsah stránek, které jsou složeny pomocí *PHP* souboru. Pokud v nějakém odkazu nalezneme „id=“ nebo „media=“, vypustíme celou tuto část od posledního lomítka, abychom ji následně mohli nahradit novým a správným názvem. *BeautifulSoup* nám změní kódování tohoto

souboru na *UTF-8*, proto k zapsání tohoto souboru budeme využívat toto kódování. Znovu si tak otevřeme zpracovávaný soubor a celý ho přepíšeme novými značkami.

Po nahrazování odkazů se pokusíme o další výměny. V prvé řadě se bude jednat o úpravu těch cest k souborům, které začínají lomítkem a neudávají nám přesnou cestu k souborům, které nemusí být uloženy na jakémkoliv serveru. Pokud tak existuje rozdíl v pozici zpracovávaného souboru vůči indexovému souboru upravovaného roku konference, nahradíme toto lomítko počtem skoků uloženými v *dirs*. Jestliže je pozice těchto souborů na stejné úrovni, lomítko pouze vypustíme. Následně si soubor opět přepíšeme a uložíme. Druhým krokem je náhrada lomítka v odkazu. Víme, že odkaz je zapsán v uvozovkách, a proto pokud nalezneme takovou sekvenci znaků, tak k tomuto lomítku pouze připíšeme *index.html* (pokud bychom tento odkaz spouštěli na serveru, odkaz zakončený lomítkem zobrazí automaticky soubor *index.html*, v našem případě není vždy povinností tento úplný web mít nahrán na serveru). Posledním krokem k statickému využití webů je nahrazení koncovky *.php*. *PHP* soubory opět nemůžeme spouštět na stroji bez serveru, a jelikož jsme se postarali o ukládání těchto souborů v podobě *HTML* dokumentů, provedeme pouze náhradu takové koncovky. Všechny provedené kroky jsou viditelné pro svou délku v příloze C.

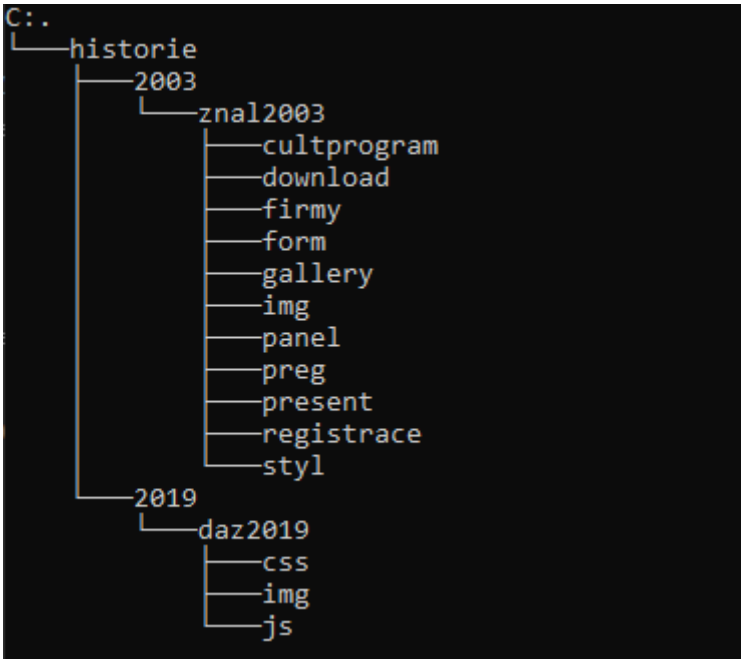
5.2.8 Průběh stahování a finální podoba webu

V tuto chvíli máme připravené veškeré skripty pro zpracování webových stránek. Nám tak jen stačí zvolit stahovaný index konference, jeho ročník a vyčkat, než budou veškeré dokumenty staženy (obrázek č. 10).

```
soubor: historie/2019/daz2019/js/jquery.easing.min.js
soubor: historie/2019/daz2019/js/scripts.js
soubor: historie/2019/daz2019/js/jquery-2.1.4.min.js
soubor: historie/2019/daz2019/js/bootstrap.min.js
Probíhá úprava odkazů a generování indexu konference...
úprava souboru: /2003/znal2003/accepted.html
úprava souboru: /2003/znal2003/authorinstr.html
úprava souboru: /2003/znal2003/camera.html
úprava souboru: /2003/znal2003/contents.html
úprava souboru: /2003/znal2003/index.html
úprava souboru: /2003/znal2003/intro.html
```

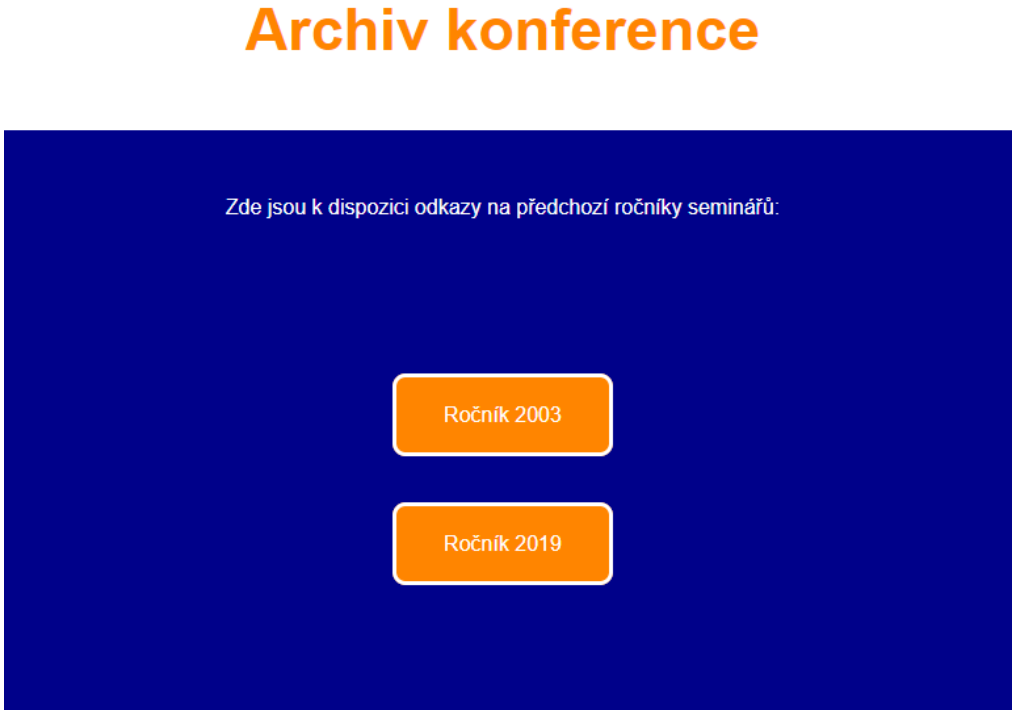
Obrázek č. 10: *Průběh stahování webových stránek*

Na disku se nám tak objeví stažené statické stránky konference, kdy na obrázku č. 11 vidíme výpis stažených složek, veškerý stažený obsah je však viditelný v příloze B.



Obrázek č. 11: *Výpis složek*

Poslední částí je zobrazení vygenerovaného indexu s odkazy na historické ročníky - v ukázce této bakalářské práce se jedná o roky 2003 a 2019 (obrázek č. 12).



Obrázek č. 12: *Index archivu konference*

Závěr

Cílem této bakalářské práce bylo prozkoumat možnosti generování statických webových stránek. Abychom však mohli samotné webové stránky pochopit, museli jsme popsat *XML* a *XSL* transformace, na kterých stojí dnešní vytváření webů. Bylo zde také popsáno, jak obdobné dokumenty vypadají a také, jaké jsou jejich historické milníky (Kapitola 1). Dále jsme popsali systém *World Wide Web*, díky kterému můžeme získávat informace o všemožných tématech. Taktéž jsme si rozlišili často zaměňované pojmy *WWW* a internet (Kapitola 2).

Třetí kapitola objasňuje svět webových stránek. Nachází se v ní popis dvou největších skupin stránek – dynamické a statické. Pro každou skupinu jsme popsali jejich základní vlastnosti. Víme tak, že pro funkci těchto stránek potřebujeme rozdílné komponenty, a že pro funkci statických stránek nepotřebujeme žádné serverové vlastnosti v porovnání se stránkami dynamickými. V druhé části této kapitoly jsme také přiblížili nejdůležitější prvky vytváření a fungování statických stránek – *HTML*, *CSS* a *JavaScript*. U každého jazyka tak byla zmíněna jeho historie, důležité milníky či osoby a také struktura jejich dokumentů.

Čtvrtá kapitola se zaobírá tvorbou webových stránek. Zjistili jsme, jak je možné vygenerovat stránky dynamické a stránky statické. U dynamických stránek jsme popsali pojem *Content Managment System* a k nim dva nejznámější systémy, u kterých jsme přehledně popsali jejich historii. Pro tvorbu statických stránek jsme popsali tvorbu pomocí generátoru. Bylo zmíněno, že takových systému existují stovky, avšak všechny se drží šesti základních bodů, které jsou v práci blíže popsány - jednotný programovací jazyk, možnost několika template jazyků, práce pomocí příkazového řádku, ukládání dat v textovém souboru, využití příkazového řádku a také rozšiřitelnost. V další části této kapitoly jsme se seznámili s nejznámějšími generátory, jejich vlastnostmi a také zde byla větší pozornost věnována generátoru *Pelican*, který jsme využívali v praktické tvorbě webových stránek. Zmínili jsme tak jeho základní vlastnosti, popsali jsme jazyk *Markdown*, vysvětlili jsme samotnou strukturu generátoru, generování vzhledu a v neposlední řadě jeho vychytávky. Na konci se nachází přehledná tabulka pro porovnání generátorů *Jekyll*, *Hugo*, *Next.js* a *Pelican*.

Poslední kapitola vysvětluje tvorbu samotných statických stránek. V první části jsme popisovali vytváření stránek o *Programování I.* Petra Šalouna. Ukázali jsme, jakým způsobem se tvoří obsah, statické cesty a také to, jak můžeme přidat vyhledávací prvek. V druhé části této kapitoly jsme se zaobírali jinou tvorbou statických webových stránek, a to pomocí stahování stránek z internetu. Samotným stažením a následnou úpravou odkazů jsme zajistili lokální statické stránky s generováním přehledného indexu pro odkazování na všechny stažené konference.

Při tvoření stránek a jejich stahování jsme zjistili, že se jedná o náročný proces. U prvního případu – statického generátoru – je tvorba celkem snadná, problém ale může nastat v případě, kdy budeme potřebovat provést mnoho aktualizací již předem vytvořeného obsahu. Může se jednat o časově náročný proces a neexistuje žádná kontrola případných chyb. V druhém případě jsme ukázali, že stahování webu nemusí být náročné. Pokud bychom však na delší časový úsek chtěli stahovat mnoho archivních ročníků, nemusíme být dynamičtí vůči stahovaným datům. Samotný skript by potřeboval spoustu úprav, aby byl vhodný i pro velmi staré weby nebo pro nové stránky, u kterých vývoj nadále pokračuje.

Literatura

[1] **W3C.** Extensible Markup Language (XML) 1.0 (Fifth Edition). [Online] 2006. [Citace: 5. květen 2020.] https://www.w3.org/TR/xml/.

[2] **W3C**. Extensible Markup Language (XML) 1.1 (Second Edition). [Online] 2006. [Citace: 5. 5 2020.] https://www.w3.org/TR/xml11/.

[3] **Kosek, Jiří.** *XML pro každého: podrobný průvodce.* Praha : Grada, 2000. ISBN 80-716-9860-1.

[4] **Pichlik, Roman.** XSLT, svět transformací. [Online] 2003. [Citace: 2. březen 2020.] http://www.sovavsiti.cz/2003/xslt-1.html.

[5] **W3C.** XSL Transformations (XSLT) Version 2.0 (Second Edition). [Online] 2009. [Citace: 2. březen 2020.] https://www.w3.org/TR/2009/PER-xslt20-20090421/.

[6] **W3C**. XSL Transformations (XSLT) Version 3.0. [Online] 2017. [Citace: 2. březen 2020.]

[7] **Kosek, Jiří.** XSLT v příkladech, Impementace. [Online] 2003. [Citace: 23. únor 2020.] https://www.kosek.cz/xml/xslt/implementace.html.

[8] **W3C.** XSL Transformations (XSLT) Version 1.0. [Online] 1999. [Citace: 1. březen 2020.] https://www.w3.org/TR/1999/REC-xslt-19991116.

[9] **Foundation, World Wide Web.** History of the Web. [Online] [Citace: 20. únor 2020.] https://webfoundation.org/about/vision/history-of-the-web/.

[10] **Computer Hope.** URL. [Online] 2019. [Citace: 20. únor 2020.] https://www.computerhope.com/jargon/u/url.htm.

[11] **Schools, W3C.** What is HTTP? [Online] [Citace: 20. únor 2020.] https://www.w3schools.com/whatis/whatis_http.asp.

[12] **Javatpoint.** What is World Wide Web. [Online] [Citace: 20. únor 2020.] https://www.javatpoint.com/what-is-world-wide-web.

[13] **W3.** Definition of Open Standards. [Online] 2005. [Citace: 2. březen 2020.] https://www.w3.org/2005/09/dd-osd.html.

[14] **Pixabey.** Stunning free images & royalty free. [Online] [Citace: 1. únor 2020.] https://pixabay.com/.

[15] **Woodford, Chris.** How the World Wide Web (WWW) works. [Online] 2018. [Citace: 2. březen 2020.] https://www.explainthatstuff.com/howthewebworks.html.

[16] **Laurenčík, Marek.** *Tvorba www stránek v HTML a CSS.* Praha : Grada, 2019. ISBN 978-80-271-2241-7.

[17] **Mozilla.** What is the difference between webpage, website, web server, and search engine? [Online] 2019. [Citace: 2. březen 2020.] https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Pages_sites_servers_and_search_engines.

[18] **Raymond, Camden a Brian, Rinaldi.** *Working with static sites: bringing the power of simplicity to modern sites.* Sebastopol, California : O'Reilly, 2017. 978-149-1960-943.

[19] **Randby, Scott P.** Dynamic and Static Website Security. [Online] 2019. [Citace: 5. březen 2020.] https://srandby.org/digital-writing/index.html.

[20] **Computer Hope.** Dynamic website. [Online] 2018. [Citace: 6. březen 2020.] https://www.computerhope.com/jargon/d/dynasite.htm.

[21] **Doteasy.** What is a Dynamic Web Page? [Online] [Citace: 7. březen 2020.] https://www.doteasy.com/web-hosting-articles/what-is-a-dynamic-web-page.cfm.

[22] **Amelia.** Static vs Dynamic Website: What Is the Difference? [Online] 2019. [Citace: 10. březen 2020.] https://wpamelia.com/static-vs-dynamic-website/.

[23] **WHATWG.** HTML Living Standard. [Online] 2020. [Citace: 13. brezen 2020.]

[24] **Staníček, Petr.** *CSS Kaskádové styly: kompletní průvodce.* Praha : Computer Press, 2003. 80-722-6872-4.

[25] **Haverbeke, Marijn.** *Eloquent JavaScript: a modern introduction to programming, Third edition.* San Francisco, California : No Starch Press, 2019. ISBN 15-932-7950-7.

[26] **Negrino, Tom a Smith, Dori.** *JavaScript pro World Wide Web.* Praha : SoftPress, 2001. 80-864-9709-7.

[27] **Kinsta.** What Is a Content Management System (CMS)? [Online] 2020. [Citace: 1. duben 2020.] https://kinsta.com/knowledgebase/content-management-system/.

[28] **Drupal.** O Drupalu. [Online] [Citace: 2. duben 2020.] https://www.drupal.cz/o-drupalu/.

[29] **Kinsta.** The History of WordPress, its Ecosystem and Community. [Online] 2019. [Citace: 1. duben 2020.] https://kinsta.com/learn/wordpress-history/.

[30] **Rinaldi, Brian.** *Static Site Generators: Modern Tools for Static Website Development.* Sebastopol, California : O'Reilly, 2016. ISBN 978-1-491-92662-8.

[31] **Jekyll.** Dokumentace. [Online] [Citace: 1. březen 2020.] https://jekyllrb.com/docs/.

[32] **Hugo.** Dokumentace. [Online] [Citace: 2. březen 2020.] https://gohugo.io/documentation/.

[33] **Next.js.** Getting started. [Online] [Citace: 2. březen 2020.] https://nextjs.org/docs/getting-started/.

[34] **Hyde.** Website. [Online] [Citace: 2. březen 2020.] http://hyde.github.io/.

[35] **Pelican.** Dokumentace. [Online] [Citace: 2. březen 2020.] https://docs.getpelican.com/en/stable/.

[36] **Gruber, John.** Markdown. [Online] 2004. [Citace: 9. březen 2020.] https://daringfireball.net/projects/markdown/.

[37] **GitHub.** Mastering Markdown. [Online] 2014. [Citace: 10. březen 2020.] https://guides.github.com/features/mastering-markdown/.

[38] **Pallets.** Jinja 2.11. [Online] [Citace: 2. březen 2020.] https://jinja.palletsprojects.com/en/2.11.x/.

[39] **PyPA.** pip - The Python Package Installer. [Online] [Citace: 6. březen 2020.] https://pip.pypa.io/en/stable/ .

[40] **Tipue.** Tipue Search. [Online] [Citace: 5. duben 2020.] https://tipue.com/search/.

[41] **Guru99.** Python Internet Access using Urllib.Request and urlopen(). [Online] [Citace: 5. duben 2020.] https://www.guru99.com/accessing-internet-data-with-python.html.

[42] **Crummy.** Beautiful Soup Documentation. [Online] [Citace: 5. duben 2020.] https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[43] **PythonForBeginners.** Python's OS Module. [Online] 2013. [Citace: 5. duben 2020.] https://www.pythonforbeginners.com/os/pythons-os-module.

[44] **techopedia.** Query String. [Online] 2017. [Citace: 29. březen 2020.] https://www.techopedia.com/definition/1228/query-string/.

[45] **W3C.** XSL Transformations (XSLT) Version 1.0. [Online] 1999. [Citace: 2. březen 2020.] https://www.w3.org/TR/1999/REC-xslt-19991116/.

Přílohy

Seznam příloh

Označení přílohy	Název přílohy
A	Seznam přiložených souborů
B	Obsah složky historie (všechny stažené soubory)
C	Skript souboru soubory.py
D	Ukázka vygenerovaného webu
E	Uživatelská příručka

Příloha A: *Seznam přiložených souborů k bakalářské práci*

Statické stránky za využití SSG Pelican

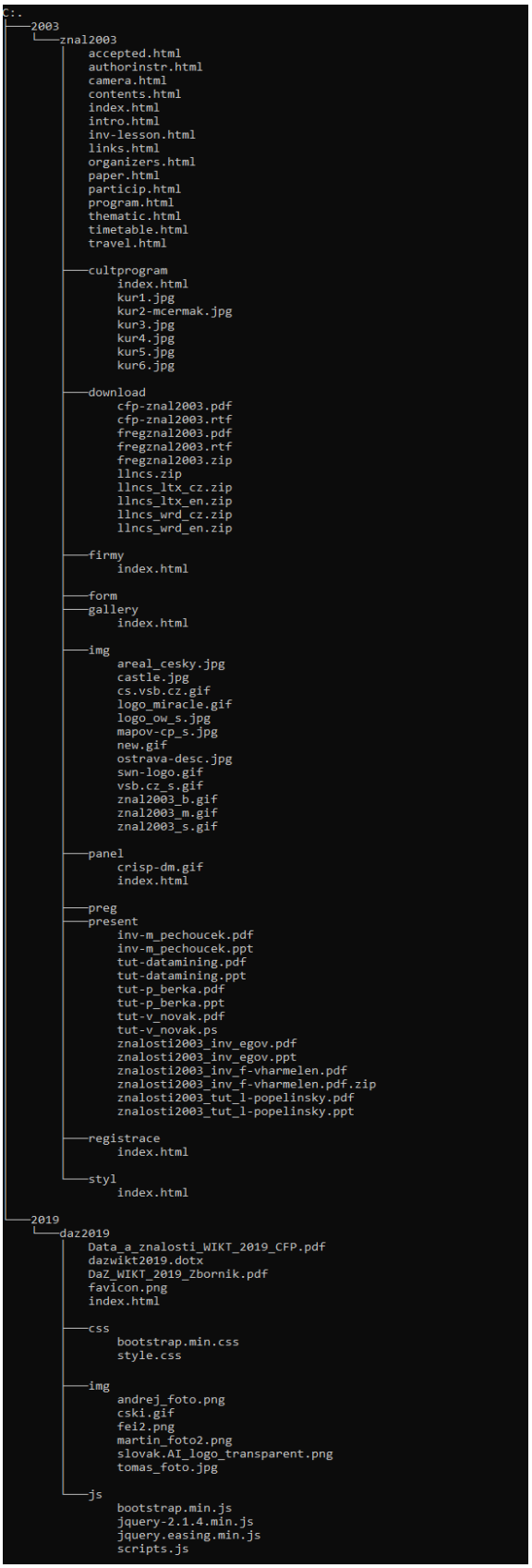
- Složka *content* – obsahuje veškeré obsahové soubory *Markdown*.
- Složka *output* – obsahuje vygenerované webové stránky.
- Upravený soubor *pelicanconf.py*.
- Složka *theme* – obsahuje upravované soubory *base.html* a *search.html*.

Stahování statických stránek

- Složka *historie* – obsahuje dva ukázkově stažené ročníky *2019* a *2003*.
- Soubor *all_links.py* – obsahuje v sobě dva listy s odkazy na stažené ročníky.
- Soubor *main.py* – obsahuje v sobě veškeré potřebné funkce pro stažení, pomocí tohoto souboru se samotný skript spouští.
- Soubor *soubory.py* – má za úkol upravování stažených *HTML* souborů.

Všechny přiložené soubory jsou dostupné také na webové adrese: http://vav0188.wz.cz/. Na této adrese se nachází také úplná složka *content*, která v sobě ještě ukrývá podložky *pdf*, *img* a *files* využívaných pro generování stránek. V této části také ukrývají vygenerované soubory *publishconf.py, tasks.py* a *makefile.* V neposlední řadě je zde úplná vzhledová složka *theme* (http://vav0188.wz.cz/pelican.zip). Na této adrese se také nacházejí vygenerované stránky *Programování I* (http://vav0188.wz.cz/pelican/index.html) a *Archiv konferencí* (http://vav0188.wz.cz/konference/index.html). Potřebné soubory pro s*tahování statických stránek* jsou taktéž zde – http://vav0188.wz.cz/skripty.zip.

Příloha B: *Obsah složky historie (všechny stažené soubory)*



Příloha C: *Skript souboru soubory.py*

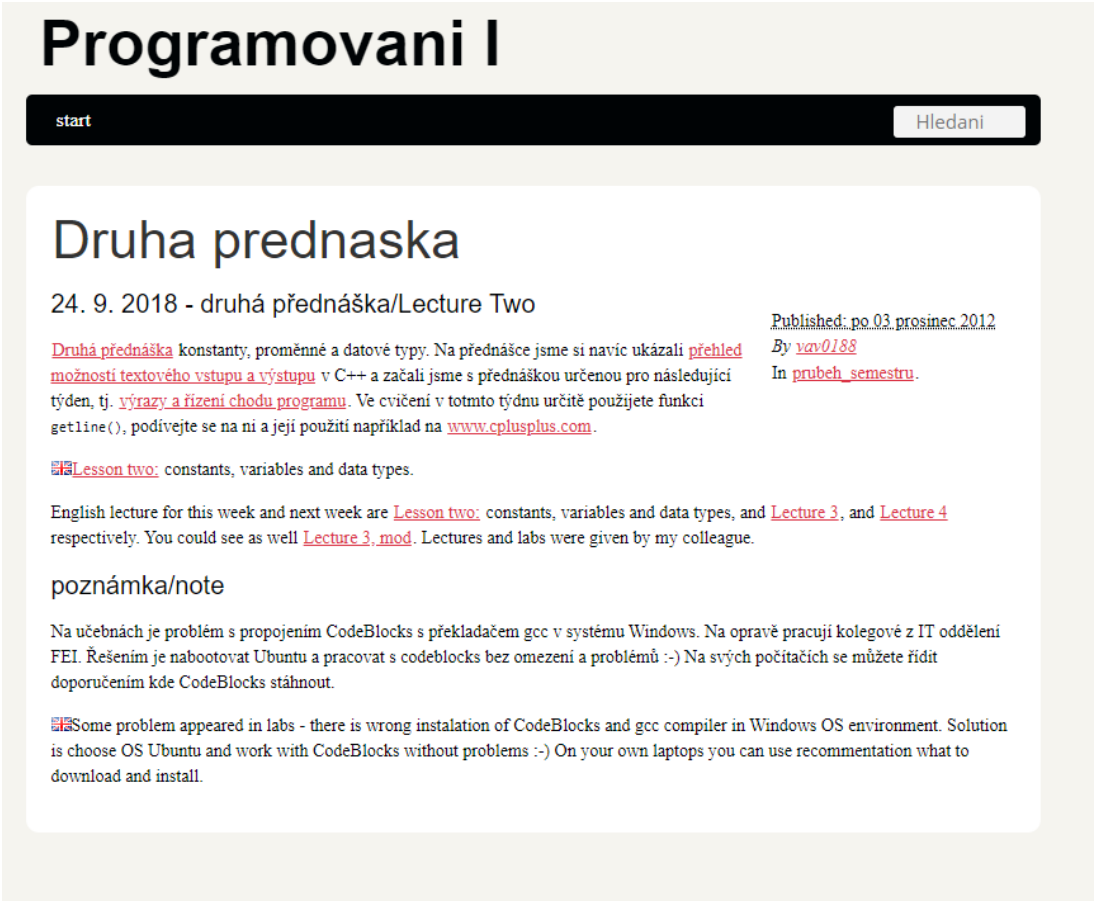
```
directory = os.getcwd()+r"\historie"
directory.replace("\\", "/")
files = []
os.chdir(directory)
for i,j,k in os.walk('.'):
    pos = i.find('.')
    dire = i[pos+1:].replace("\\", "/")
    path = directory+dire
    os.chdir(path)
    for file in glob.glob("*.html"):
        files.append(path+"/"+file)
    os.chdir(directory)
cnt = len(directory)
for fl in files:
    f = fl[cnt:]
    frs = f.find("/")
    scd = f[frs+1:].find("/")
    year = f[frs+1:scd+1]
    pos = -1
    for i in links2:
        pos = pos + 1
        if i.find(year) > 0:
            break
    dirs = ""
    dr = ""
    url = links2[pos]
    first = url.find("/")
    second = url.find("/", first + 1)
    count = url[second+1:].count("/")
    for i in range(0, abs(count)):
        dr += "../"
    print("úprava souboru: " + f)
    first = f.find("/")
    second = f.find("/", first + 1)
    count2 = f[second + 1:].count("/")
    count3 = count2 - count
    for i in range(0, abs(count3)):
        dirs += "../"
    txt = ""
    cod = ""
    with open(fl, 'rb') as inf:
        cod = detect(inf.read())['encoding']
    if cod == "ISO-8859-1":
        cod = 'iso-8859-2'
    with open(fl, 'r',encoding=cod, errors="ignore") as inf:
```

```
        txt = inf.read()
    soup = BeautifulSoup(txt,'html.parser')
    all_a = soup.findAll('a')
    changed = False
    for a in all_a:
        for i in range(0, len(links)):
            if a.get('href') != None:
                href = a['href']
                if href.find(links[i]) > -1:
                    a['href'] = dr+dirs+links2[i]
                    changed = True
                posId = href.find("id=")
                posMed = href.find("media=")
                if posId > -1 or posMed > -1:
                    if href.find("&") > -1:
                        posAnd = href.find("&", posId + 3)
                        if posId > -1:
                            a['href'] = href[:href.rfind("/") + 1] +
href[posId + 3:posAnd] + ".html"
                            changed = True
                        elif posMed > -1:
                            a['href'] = href[:href.rfind("/") + 1] +
href[posMed + 6:posAnd]
                            changed = True
                    else:
                        if posId > -1:
                            a['href'] = href[:href.rfind("/") + 1] +
href[posId + 3:] + ".html"
                            changed = True
                        elif posMed > -1:
                            a['href'] = href[:href.rfind("/") + 1] +
href[posMed + 6:]
                            changed = True
        base = soup.findAll('base')
        for b in base:
            changed == True
            b.decompose()
        charenc = "utf-8"
        if changed == True:
            with open(fl, 'r+', encoding=charenc, errors='ignore')
as file:
                file.seek(0)
                file.truncate()
                file.write(str(soup))
                file.close()
            charenc = "utf-8"
        else:
```

```
        charenc = cod
with open(fl, 'r+', encoding=charenc,errors='ignore') as file:
    content = file.read()
    file.seek(0)
    file.truncate()
    if count3 > 0:
        file.write(content.replace('="/', '="'+dirs))
    else:
        file.write(content.replace('="/', '="'))
    file.close()
with open(fl, 'r+', encoding=charenc,errors='ignore') as file:
    content = file.read()
    file.seek(0)
    file.truncate()
    file.write(content.replace('/"', '/index.html"'))
    file.close()
with open(fl, 'r+', encoding=charenc,errors='ignore') as file:
    content = file.read()
    file.seek(0)
    file.truncate()
    file.write(content.replace('.php', '.html"'))
    file.close()
```

Příloha D: *Ukázka vygenerovaného webu*

Obrázek D.1: *Stránka search.html*



Obrázek D.2: *Ukázka průběhu semestru - druha-prednaska.html*

Příloha E: *Uživatelská příručka*

Tato uživatelská příručka se vztahuje k vytvořenému softwaru v rámci této bakalářské práce (XML/XSLT systém pro lokální generování statických webových stránek, Pavel Vavruška, VŠB - TUO Katedra informatiky). Hlavním úkolem tohoto programu je stahování webových stránek, včetně jejich multimediálních souborů, skriptů či stylů. Druhá část tohoto software je zaměřena na upravování samotných stažených *HTML* dokumentů. Úkolem této části je upravit odkazy v těchto dokumentech, případně provést propojení jednotlivých stažených ročníků.

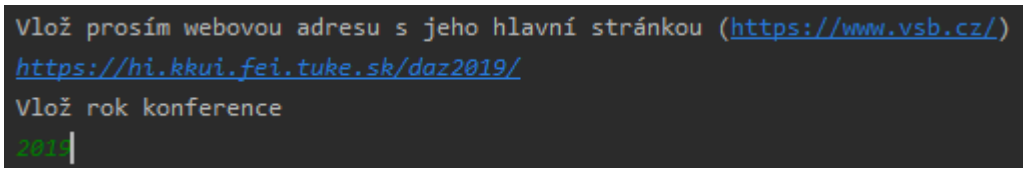
Pro spuštění tohoto software je důležité splnit několik podmínek:

- Je nutné mít nainstalován *Python 3* a jeho moduly *BeautifulSoup*, *urllib* a *chardet*.
- Je nutné mít skripty *main.py*, *soubory.py* a *all_links.py* v jedné složce, ve které se budou stránky generovat.
- Skript zvládá stahovat a upravovat *HTML* a *PHP* soubory.
- Skript stahuje veškeré soubory uloženy ve zpracovávaném *HTML* nebo *PHP* dokumentu.
- Je nezbytné nezasahovat do skriptu *all_links.py*, uchovává v sobě nahrazované odkazy ročníků, které jsou staženy do složky *historie*.

Spuštění software

V tuto chvíli stačí otevřít skript *main.py* pomocí *Pythonu*. Posléze stačí odpovědět na dvě otázky – odkaz na stahovaný web a následně rok zadané konference.

Pro první otázku je nezbytné dodržet správný formát. Jako odkaz musí být zadán ten, který se považuje za hlavní stránku stahované konference, jako je například https://daz2017.kiv.zcu.cz/. Pokud by se zvolila stránka jiná, například podstránka pro fotografie (pro ukázku https://daz2017.kiv.zcu.cz/fotografie/), nemusel by skript fungovat správně kvůli ověřování správnosti domény, bude si totiž myslet, že složka *fotografie* je nezbytnou součástí pro stahování. Je proto povinné zvolit stránku hlavní. U druhé otázky pro rok konference je pouze nutnost zvolit rok v rozmezí 1900 po současný rok. V případě špatně zadaných parametrů umožní software jejich opravení.



Obrázek E.1: *Uživatelský vstup*

Ve chvíli, kdy jsou všechny informace zadány, skript začíná web stahovat. Doporučuje se tedy čekat. Program po stažení oznámí, že bude probíhat nahrazování a upravování odkazů pro správné použití. Po upravení odkazů se otevře indexová stránka se odkazy na všechny stažené konference.

```
soubor: historie/2019/daz2019/js/jquery.easing.min.js
soubor: historie/2019/daz2019/js/scripts.js
soubor: historie/2019/daz2019/js/jquery-2.1.4.min.js
soubor: historie/2019/daz2019/js/bootstrap.min.js
Probíhá úprava odkazů a generování indexu konference...
úprava souboru: /2003/znal2003/accepted.html
úprava souboru: /2003/znal2003/authorinstr.html
úprava souboru: /2003/znal2003/camera.html
úprava souboru: /2003/znal2003/contents.html
úprava souboru: /2003/znal2003/index.html
úprava souboru: /2003/znal2003/intro.html
```

Obrázek E.2: *Průběh stahování a nahrazování odkazů*

Vygenerovaný index může vypadat například takto:

Obrázek E.3: *Vygenerovaný index.html*